2 PROGRAMMING INTERFACE

In this chapter, the internal operations of the Cyrix III CPU are described mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register set, memory addressing, various types of interrupts and the shutdown and halt process. An overview of real, virtual 8086, and protected operating modes is also included in this chapter. The FPU operations are described separately at the end of the chapter.

2.1 Processor Initialization

The Cyrix III CPU is initialized when the RESET# signal is asserted. The processor is placed in real mode and the registers listed in Table 2-1 (Page 2-16) are set to their initialized values. RESET# invalidates and disables the cache and turns off paging. When RESET# is asserted, the Cyrix III CPU terminates all local bus activity and all internal execution. During the entire time that RESET# is asserted, the internal pipelines are flushed and no instruction execution or bus activity occurs.

Approximately 150 to 250 external clock cycles after RESET# is negated, the processor begins executing instructions at the top of physical memory (address location FFFF FFF0h). Typically, an intersegment JUMP is placed at FFFF FFF0h. This instruction will force the processor to begin execution in the lowest 1 MB of address space.

Note: The actual time depends on the clock scaling in use. Also an additional 2^{20} clock cycles are needed if self-test is requested.

Table 2-1. Initialized Core Registers Contents

Register	Register Name	Initialized Contents	Comments
EAX	Accumulator	xxxx xxxxh	0000 0000h indicates self-test passed.
EBX	Base	xxxx xxxxh	
ECX	Count	xxxx xxxxh	
EDX	Data	xxxx 04 [DIR0]	DIR0 = Device ID
EBP	Base Pointer	xxxx xxxxh	
ESI	Source Index	xxxx xxxxh	
EDI	Destination Index	xxxx xxxxh	
ESP	Stack Pointer	xxxx xxxxh	
EFLAGS	Flags	0000 0002h	See Table 2-6 on page 2-26 for bit definitions.
EIP	Instruction Pointer	0000 FFF0h	
ES	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
CS	Code Segment	F000h	Base address set to FFFF 0000h. Limit set to FFFFh.
SS	Stack Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
DS	Data Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
FS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
GS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
IDTR	Interrupt Descriptor Table Register	Base = 0, Limit = 3FFh	
GDTR	Global Descriptor Table Register	xxxx xxxxh xxxxh	
LDTR	Local Descriptor Table Register	xxxx xxxxh, xxxxh	
TR	Task Register	xxxxh	
CR0	Machine Status Word	6000 0010h	See Table 2-12 on page29 for bit definitions.
CR2	Control Register 2	xxxx xxxxh	See page30.
CR3	Control Register 3	xxxx xxxxh	See page30.
CR4	Control Register 4	0000 0000h	See Table 2-9 on page 2-30 for bit definitions.
CCR1	Configuration Control 1	00h	See paragraph 2.5.4.1 on page52 for bit definitions.
CCR2	Configuration Control 2	00h	See paragraph 2.5.4.3 on page53 for bit definitions.
CCR3	Configuration Control 3	00h	See paragraph 2.5.4.4 on page54 for bit definitions.
CCR7	Configuration Control 7	00h	See paragraph 2.5.4.8, page58 for bit definitions.
DIR0	Device Identification 0	4xh	Device ID and reads back initial CPU clock-speed setting.
DIR1	Device Identification 1	xxh	Stepping and Revision ID (RO).
DR7	Debug Register 7	0000 0400h	See (XREF) for bit definitions.

Table 2-1. Initialized Core Registers Contents (Continued)

Register	Register Name	Initialized Contents	Comments

x = Undefined value

2.2 Instruction Set Overview

The Cyrix III Processor instruction set can be divided into ten types of operations:

- Arithmetic
- Shift/Rotate
- Control Transfer
- Data Transfer
- Floating Point
- High-Level Language Support
- Operating System Support
- Bit Manipulation
- String Manipulation
- MMX and 3DNow! Instructions

Cyrix III Processor instructions operate on as few as zero operands and as many as three operands. An NOP instruction (no operation) is an example of a zero-operand instruction. Two-operand instructions allow the specification of an explicit source and destination pair as part of the instruction. These two operand instructions can be divided into eight groups according to operand types:

- Register to Register
- Register to Memory
- Memory to Register
- Memory to Memory
- Register to I/O
- I/O to Register
- Immediate Data to Register
- Immediate Data to Memory

An operand can be held in the instruction itself (as in the case of an immediate operand), in one of the processor's registers or I/O ports, or in memory. An immediate operand is fetched as part of the opcode for the instruction.

Operand lengths of 8, 16, 32 or 48 bits are supported as well as 64 or 80 bits associated with floating-point instructions. Operand lengths of 8 or 32 bits are generally used when executing code written for 386- or 486-class (32-bit code) processors. Operand lengths of 8 or 16 bits are generally used when executing existing 8086 or 80286 code (16-bit code). The default length of an operand can be overridden by placing one or more instruction prefixes in front of the opcode. For example, the use of prefixes allows a 32-bit operand to be used with 16-bit code or a 16-bit operand to be used with 32-bit code.

Chapter 6 of this manual lists each instruction in the Cyrix III CPU instruction set along with the associated opcodes, execution clock counts, and effects on the FLAGS register.

2.2.1 Lock Prefix

The LOCK prefix may be placed before certain instructions that read, modify, then write back to memory. The LOCK prefix can be used with the following instructions only when the result is a write operation to memory:

Bit Test Instructions (BTS, BTR, BTC)
Exchange Instructions (XADD, XCHG, CMPXCHG)
One-operand Arithmetic and Logical Instructions (DEC, INC, NEG, NOT)
Two-operand Arithmetic and Logical Instructions (ADC, ADD, AND, OR, SBB, SUB, XOR).

An invalid opcode exception is generated if the LOCK prefix is used with any other instruction or with one of the instructions above when no write operation to memory occurs (for example, when the destination is a register).

2.3 Register Sets

From the programmer's point of view the accessible registers in the Cyrix III CPU are grouped into two sets of registers, the application and system registers set. The application register set contains the registers frequently used by application programmers, and the system register set contains the registers typically reserved for use by operating system programmers.

The application register set is made up of general purpose registers, segment registers, a flag register, and an instruction pointer register.

The system register set is made up of the remaining registers which include control registers, system address registers, debug registers, configuration registers, and test registers.

Each of the registers is discussed in detail in the following sections.

2.3.1 Application Register Set

The Application Register Set, as shown in Table 2-2, consists of the registers most often used by the applications programmer.

These registers are generally accessible, although some bits in the Flags register are protected.

The **General Purpose Register** contents are frequently modified by instructions and typically contain arithmetic and logical instruction operands.

In real mode, **Segment Registers** contain the base address for each segment. In protected mode, the segment registers contain segment selectors. The segment selectors provide indexing for tables (located in memory) that contain the base address for each segment, as well as other memory addressing information.

The **Instruction Pointer Register** points to the next instruction that the processor will execute. This register is automatically incremented by the processor as execution progresses.

The **Flags Register** contains control bits used to reflect the status of previously executed instructions. This register also contains control bits that affect the operation of some instructions.

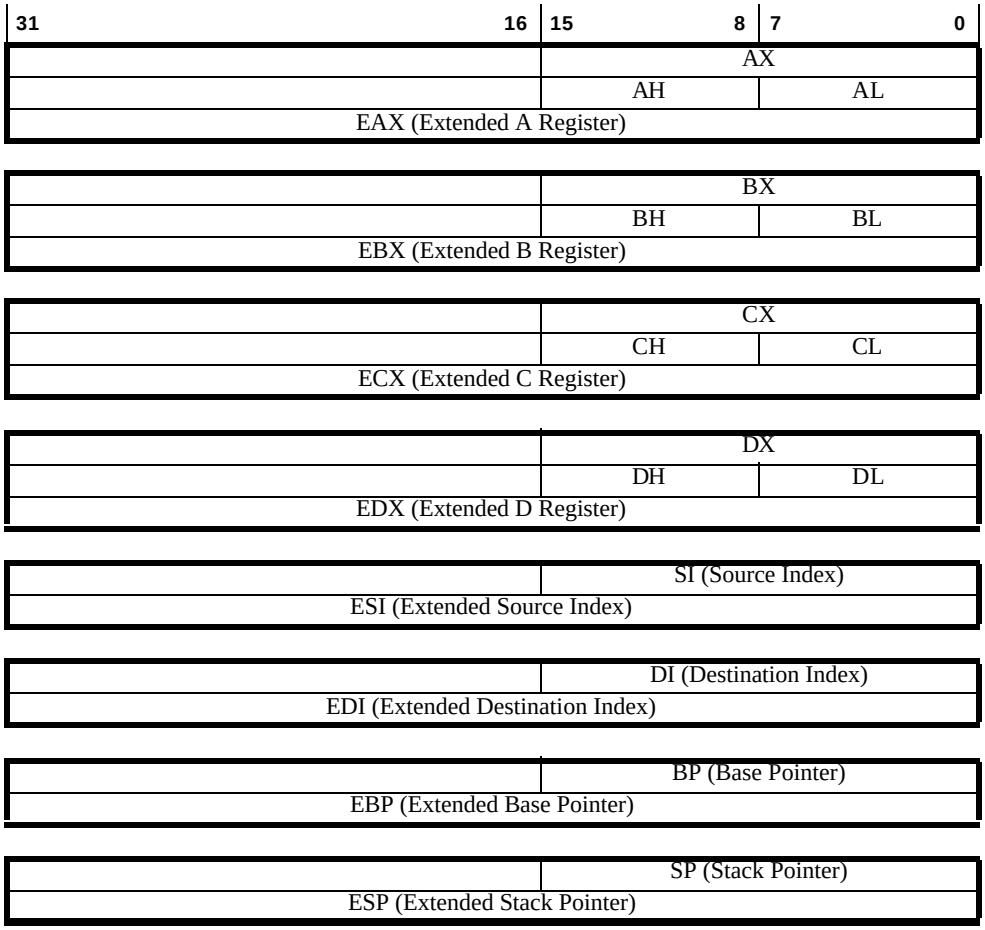
2.3.2 General Purpose Registers

The General Purpose Registers are divided into four data registers, two pointer registers, and two index registers as shown in Table 2-2 on page 2-20.

The **Data Registers** are used by the applications programmer to manipulate data structures and to hold the results of logical and arithmetic operations. Different portions of a general data registers can be addressed by using different names. An “E” prefix identifies a complete 32-bit register. An “X” suffix without the “E” prefix identifies the lower 16 bits of the register.

The lower two bytes of a data register are addressed with an “H” suffix (identifies the upper byte) or an “L” suffix (identifies the lower byte). These _L and _H portions of the data registers act as independent registers. For example, if the AH register is written to by an instruction, the AL register bits remain unchanged.

Table 2-2. Application Register Set
General Purpose Registers



The **Pointer and Index Registers** are listed below.

SI or ESI	Source Index
DI or EDI	Destination Index
SP or ESP	Stack Pointer
BP or EBP	Base Pointer

These registers can be addressed as 16- or 32-bit registers, with the "E" prefix indicating 32 bits.

The pointer and index registers can be used as general purpose registers; however, some instructions use a fixed assignment of these registers. For example, repeated string operations always use ESI as the source pointer, EDI as the destination pointer, and ECX as a counter. The instructions that use fixed registers include multiply and divide, I/O access, string operations, stack operations, loop, variable shift and rotate, and translate instructions.

The Cyrix III Processor implements a stack using the ESP register. This stack is accessed during the PUSH and POP instructions, procedure calls, procedure returns, interrupts, exceptions, and interrupt/exception returns. The Cyrix III Processor automatically adjusts the value of the ESP during operations that result from these instructions.

The EBP register may be used to refer to data passed on the stack during procedure calls. Local data may also be placed on the stack and accessed with BP. This register provides a mechanism to access tack data in high-level languages.

2.3.3 Segment Registers and Selectors

Segmentation provides a means of defining data structures inside the memory space of the microprocessor. There are three basic types of segments: code, data, and stack. Segments are used automatically by the processor to determine the location in memory of code, data, and stack references.

There are six 16-bit segment registers as shown in Table 2-3.

Table 2-3. Application Register Set Segment Selector Registers

15	0
CS (Code Segment)	
SS (Stack Segment)	
DS (D Data Segment)	
ES (E Data Segment)	
FS (F Data Segment)	
GS (G Data Segment)	

In real and virtual 8086 operating modes, a segment register holds a 16-bit segment base. The 16-bit segment is multiplied by 16 and a 16-bit or 32-bit offset is then added to it to create a linear address. The offset size is dependent on the current address size. In real mode and in virtual 8086 mode with paging disabled, the linear address is also the physical address. In virtual 8086 mode with paging enabled, the linear address is translated to the physical address using the current page tables.

Table 2-4. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment	Segment-Override Prefix
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS instructions	ES	None
Other data references with effective address using base registers of:	DS	CS, ES, FS, GS, SS
EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP	SS	CS, DS, ES, FS, GS

In protected mode a segment register holds a Segment Selector containing a 13-bit index, a Table Indicator (TI) bit, and a two-bit Requested Privilege Level (RPL) field. The Index points into a descriptor table in memory and selects one of 8192 (2^{13}) segment descriptors contained in the descriptor table.

A segment descriptor is an eight-byte value used to describe a memory segment by defining the segment base, the segment limit, and access control information. To address data within a segment, a 16-bit or 32-bit offset is added to the segment's base address. Once a segment selector has been loaded into a segment register, an instruction needs only to specify the segment register and the offset.
The Table Indicator (TI) bit of the selector defines which descriptor table the index points into. If TI=0, the index references the Global Descriptor Table (GDT). If TI=1, the index references the Local Descriptor Table (LDT). The GDT and LDT are described in more detail in Section 2.4.2 (Page 2-33). Protected mode addressing is discussed further in Sections 2.8.2 (Page 2-78).

The Requested Privilege Level (RPL) field in a segment selector is used to determine the Effective Privilege Level of an instruction (where RPL=0 indicatesthe most privileged level, and RPL=3 indicatesthe least privileged level).

If the level requested by RPL is less than the Current Program Level (CPL), the RPL level is accepted and the Effective Privilege Level is changed to the RPL value. If the level requested by RPL is greater than CPL, the CPL overrides the requested RPL and Effective Privilege Level remains unchanged.

When a segment register is loaded with a segment selector, the segment base, segment limit and access rights are loaded from the descriptor table entry into a user-invisible or hidden portion of the segment register (i.e., cached on-chip). The CPU does not access the descriptor table entry again until another segment register load occurs. If the descriptor tables are modified in memory, the segment registers must be reloaded with the new selector values by the software.

The active segment register is selected according to the rules listed in Table 2-4 and the type of instruction being currently processed. In general, the DS register selector is used for data references. Stack references use the SS register, and instruction fetches use the CS register. While some of these selections may be overridden, instruction fetches, stack operations, and the destination write operation of string operations cannot be overridden. Special segment-override instruction prefixes allow the use of alternate segment registers. These segment registers include the ES, FS, and GS registers.

2.3.4 Instruction Pointer Register

The Instruction Pointer (EIP) Register contains the offset into the current code segment of the next instruction to be executed. The register is normally incremented by the length of the current instruction with each instruction execution unless it is implicitly modified through an interrupt, exception, or an instruction that changes the sequential execution flow (for example JMP and CALL.

Table 2-5. Application Register Set
Instruction Pointer

31	0
EIP (Extended Instruction Pointer Register)	

2.3.5 Extended Flags Register

The Extended Flags Register, EFLAGS, contains status information and controls certain operations on the Cyrix III CPU microprocessor. The lower 16 bits of this register are referred to as the FLAGS register that is used when executing 8086 or 80286 code. The flag bits listed in Table 2-6.

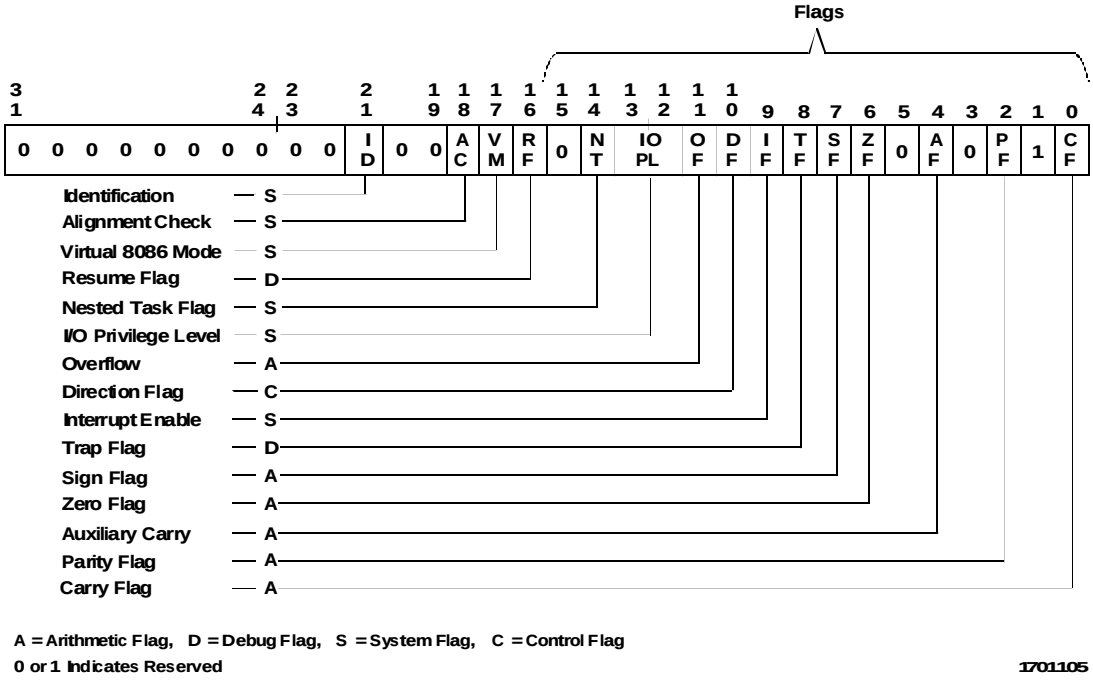


Figure 2-3. EFLAGS Register

Table 2-6. Register Bits EFLAGS Register

Bit	Name	Flag Type	Description
31:22	RSVD	--	Reserved — Set to 0.
21	ID	System	Identification Bit — The ability to set and clear this bit indicates that the CPUID instruction is supported. The ID can be modified only if the CPUID bit in CCR4 (Index E8h[7]) is set.
20:19	RSVD	--	Reserved — Set to 0.
18	AC	System	Alignment Check Enable — In conjunction with the AM flag in CR0, the AC flag determines whether or not misaligned accesses to memory cause a fault. If AC is set, alignment faults are enabled.
17	VM	System	Virtual 8086 Mode — If set while in protected mode, the processor switches to virtual 8086 operation handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set by the IRET instruction (if current privilege level is 0) or by task switches at any privilege level.
16	RF	Debug	Resume Flag — Used in conjunction with debug register breakpoints. RF is checked at instruction boundaries before breakpoint exception processing. If set, any debug fault is ignored on the next instruction.
15	RSVD	--	Reserved — Set to 0.
14	NT	System	Nested Task — While executing in protected mode, NT indicates that the execution of the current task is nested within another task.
13:12	IOPL	System	I/O Privilege Level — While executing in protected mode, IOPL indicates the maximum current privilege level (CPL) permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. IOPL also indicates the maximum CPL allowing alteration of the IF bit when new values are popped into the EFLAGS register.
11	OF	Arithmetic	Overflow Flag — Set if the operation resulted in a carry or borrow into the sign bit of the result but did not result in a carry or borrow out of the high-order bit. Also set if the operation resulted in a carry or borrow out of the high-order bit but did not result in a carry or borrow into the sign bit of the result.
10	DF	Control	Direction Flag — When cleared, DF causes string instructions to auto-increment (default) the appropriate index registers (ESI and/or EDI). Setting DF causes auto-decrement of the index registers to occur.
9	IF	System	Interrupt Enable Flag — When set, maskable interrupts (INTR input pin) are acknowledged and serviced by the CPU.
8	TF	Debug	Trap Enable Flag — Once set, a single-step interrupt occurs after the next instruction completes execution. TF is cleared by the single-step interrupt.
7	SF	Arithmetic	Sign Flag — Set equal to high-order bit of result (0 indicates positive, 1 indicates negative).
6	ZF	Arithmetic	Zero Flag — Set if result is zero; cleared otherwise.
5	RSVD	--	Reserved — Set to 0.
4	AF	Arithmetic	Auxiliary Carry Flag — Set when a carry out of (addition) or borrow into (subtraction) bit position 3 of the result occurs; cleared otherwise.
3	RSVD	--	Reserved — Set to 0.

Table 2-6. Register Bits EFLAGS Register (Continued)

Bit	Name	Flag Type	Description
2	PF	Arithmetic	Parity Flag — Set when the low-order 8 bits of the result contain an even number of ones; otherwise PF is cleared.
1	RSVD		Reserved — Set to 1.
0	CF	Arithmetic	Carry Flag — Set when a carry out of (addition) or borrow into (subtraction) the most significant bit of the result occurs; cleared otherwise.

2.4 System Register Set

The system register set is used for system level programming. The system register set consists of registers not generally used by application programmers. These registers are typically employed by system level programmers who generate operating systems and memory management programs. Associated with the system register set are tables and segments which are defined in memory.

The Control Registers control certain aspects of the Cyrix III Processor such as paging, coprocessor functions, and segment protection.

The Descriptor Tables hold descriptors that manage memory segments and tables, interrupts and task switching. The tables are defined by corresponding registers.

The two **Task State Segments Tables** defined by TSS register, are used to save and load the computer state when switching tasks.

The Configuration Registers are used to define Cyrix III Processor CPU setup including cache management.

The **ID Registers** allow BIOS and other software to identify the specific CPU and stepping. System Management Mode (SMM) control information is stored in the SMM registers.

The Debug Registers provide debugging facilities for the Cyrix III Processor and enable the use of data access breakpoints and code execution breakpoints.

The Test Registers provide a mechanism to test the contents of both the on-chip 16KB cache and the Translation Lookaside Buffer (TLB). The TLB is used as a cache for the tables that are used in to translate linear addresses to physical addresses while paging is enabled

2.4.1 Control Registers

The standard x86 Control Registers (CR0, CR2, CR3 and CR4), are shown in Table 2-7.[1] The CR0 register contains system control bits which configure operating modes and indicate the general state of the CPU. The lower 16 bits of CR0 are referred to as the Machine Status Word (MSW). The CR0 bit definitions are described in Table 2-13. The reserved bits in CR0 should not be modified. A CR1 register is not defined.

When paging is enabled and a page fault is generated, the CR2 register retains the 32-bit linear address of the address that caused the fault. When a double page fault occurs, CR2 contains the address for the second fault. Register CR3 contains the 20 most significant bits of the physical base address of the page directory. The page directory must always be aligned to a 4-KB page boundary, therefore, the lower 12 bits of CR3 are not required to specify the base address.

Register CR3 contains the 20 most significant bits of the physical base address of the page directory. The page directory must always be aligned to a 4KB page boundary, therefore, the lower 12 bits of CR3 are not required to specify the base address.

CR3 also contains the Page Cache Disable (PCD) and Page Write Through (PWT) bits. Control Register CR4 Table 2-9 on page 30 controls usage of the Time Stamp Counter Instruction, Debugging Extensions, Page Global Enable and the RDPMC instruction.

1. The CRn are standard x86 registers, and are distinct from the CCRn registers unique to Cyrix.)

Table 2-7. Control Registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CR4 Register

RSVD	PCE	PGE	RSVD	DE	TSC	RSVD

CR3 Register

PDBR (Page Directory Base Register)	RSVD	PCD	PWT	RSVD

CR2 Register

PFLA (Page Fault Linear Address)

CR1 Register

RSVD

CR0 Register

PG	CD	NW	RSVD	AM	RSVD	WP	RSVD	NE	1	TS	EM	MP	PE
							Machine Status Word (MSW)						

Table 2-8. CR4 Bit Definitions

BIT POSITION	NAME	FUNCTION
2	TSD	Time Stamp Counter Instruction If = 1 RDTSC instruction enabled for CPL=0 only; Reset State If = 0 RDTSC instruction enabled for all CPL states
3	DE	Debugging Extensions If = 1 enables I/O breakpoints and R/W bits for each debug register are defined as: 00 -Break on instruction execution only. 01 -Break on data writes only. 10 -Break on I/O reads or writes. 11 -Break on data reads or writes but not instruction fetches. If = 0 I/O breakpoints and R/W bits for each debug register are not enabled.
7	PGE	Page Global Enable If = 1 global page feature is enabled. If = 0 global page feature is disabled. Global pages are not flushed from TLB on a task switch or write to CR3
8	PCE	Performance Monitoring Counter Enable If = 1 enables execution of RDPMC instruction at any protection level. If = 0 RDPMC instruction can only be executed at protection level 0.

Table 2-9. CR3 Bit Definitions

Bits	Name	Description
31 - 12	PDBR	Page Directory Base Register: Identifies page directory base address on a 4KB page boundary.
11 - 5	RSVD	Reserved**:** Set to 0.
4	PCD	**Page Cache Disable:** During bus cycles that are not paged, the state of the PCD bit is reflected on the PCD pin. These bus cycles include interrupt acknowledge cycles and all bus cycles, when paging is not enabled. The PCD pin should be used to control caching in an external cache.
3	PWT	**Page Write-Through:** During bus cycles that are not paged, the state of the PWT bit is driven on the PWT pin. These bus cycles include interrupt acknowledge cycles and all bus cycles, when paging is not enabled. The PWT pin should be used to control write policy in an external cache.
2-1	RSVD	Reserved**:** Set to 0.

Table 2-10. CR2 Bit Definitions

Bits	Name	Description
31 - 0	PFLA	Page Fault Linear Address: With paging enabled and after a page fault, PFLA contains the linear address of the address that caused the page fault.

Table 2-11. CR1 Bit Definitions

Bit	Name	Description
31:0	RSVD	**Reserved:** Set to 0 (always returns 0 when read).

Table 2-12. CR0 Bit Definitions

Bit	Name	Description
31	PG	**Paging Enable Bit:** If PG=1 and protected mode is enabled (PE=1), paging is enabled. After changing the state of PG, software must execute an unconditional branch instruction (e.g., JMP, CALL) to have the change take effect.
30	CD	**Cache Disable:** If CD=1, no further cache line fills occur. However, data already present in the cache continues to be used if the requested address hits in the cache. Writes continue to update the cache and cache invalidations due to inquiry cycles occur normally. The cache must also be invalidated to completely disable any cache activity.
29	NW	**Not Write-Back:** If NW=1, the on-chip cache operates in write-through mode. In write-through mode, all writes (including cache hits) are issued to the external bus. If NW=0, the on-chip cache operates in write-back mode. In write-back mode, writes are issued to the external bus only for a cache miss, a line replacement of a modified line, or as the result of a cache inquiry cycle.
28:19	RSVD	**Reserved:** Do not modify.
18	AM	Alignment Check Mask: If AM=1, the AC bit in the EFLAGS register is unmasked and allowed to enable alignment check faults. Setting AM=0 prevents AC faults from occurring.
17	RSVD	**Reserved:** Do not modify.
16	WP	Write Protect: Protects read-only pages from supervisor write access. WP=0 allows a read-only page to be written from privilege level 0-2. WP=1 forces a fault on a write to a read-only page from any privilege level.
15:6	RSVD	**Reserved:** Do not modify.
5	NE	**Numerics Exception:** NE=1 to allow FPU exceptions to be handled by interrupt 16. NE=0 if FPU exceptions are to be handled by external interrupts.
4	1	**Reserved:** Do not attempt to modify.
3	TS	**Task Switched:** Set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 causes a DNA fault. If MP=1 and TS=1, a WAIT instruction also causes a DNA fault.
2	EM	**Emulate Processor Extension:** If EM=1, all floating point instructions cause a DNA fault 7.
1	MP	**Monitor Processor Extension:** If MP=1 and TS=1, a WAIT instruction causes Device Not Available (DNA) fault 7. The TS bit is set to 1 on task switches by the CPU. Floating point instructions are not affected by the state of the MP bit. The MP bit should be set to one during normal operations.
0	PE	**Protected Mode Enable:** Enables the segment based protection mechanism. If PE=1, protected mode is enabled. If PE=0, the CPU operates in real mode and addresses are formed as in an 8086-style CPU.

Table 2-13. CR0 Register EM, TS, and MP Bits Combinations

CR0 Register Bits			Instruction Type	
EM Bit 2	TS Bit 3	MP Bit 1	WAIT	ESC
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Execute	Fault 7
0	1	1	Fault 7	Fault 7
1	0	0	Execute	Fault 7
1	0	1	Execute	Fault 7
1	1	0	Execute	Fault 7
1	1	1	Fault 7	Fault 7

2.4.2 Descriptor Table Registers and Descriptors

Descriptor Table Registers

The Global, Interrupt, and Local Descriptor Table Registers (GDTR, IDTR and LDTR), shown in Figure 2-4, are used to specify the location of the data structures that control segmented memory management. The GDTR, IDTR and LDTR are loaded using the LGDT, LIDT and LLDT instructions, respectively. The values of these registers are stored using the corresponding store instructions. The GDTR and IDTR load instructions are privileged instructions when operating in protected mode. The LDTR can only be accessed in protected mode.

The Global Descriptor Table Register (GDTR) holds a 32-bit linear base address and 16-bit limit for the Global Descriptor Table (GDT). The GDT is an array of up to 8192 8-byte descriptors. When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the Local Descriptor Table (LDT) to locate a descriptor. If TI = 0, the index portion of the selector is used to locate the descriptor within the GDT table. The contents of the GDTR are completely visible to the programmer by using a SGDT instruction. The first descriptor in the GDT (location 0) is not used by the CPU and is referred to as the “null descriptor”. The GDTR is initialized using a LGDT instruction.

The Interrupt Descriptor Table Register (IDTR) holds a 32-bit linear base address and 16-bit limit for the Interrupt Descriptor Table (IDT). The IDT is an array of 256 interrupt descriptors, each of which is used to point to an interrupt service routine. Every interrupt that may occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer by using a SIDT instruction. The IDTR is initialized using the LIDT instruction.

The Local Descriptor Table Register (LDTR) holds a 16-bit selector for the Local Descriptor Table (LDT). The LDT is an array of up to 8192 8-byte descriptors. When the LDTR is loaded, the LDTR selector indexes an LDT descriptor that resides in the Global Descriptor Table (GDT). The base address and limit are loaded automatically and cached from the LDT descriptor within the GDT.

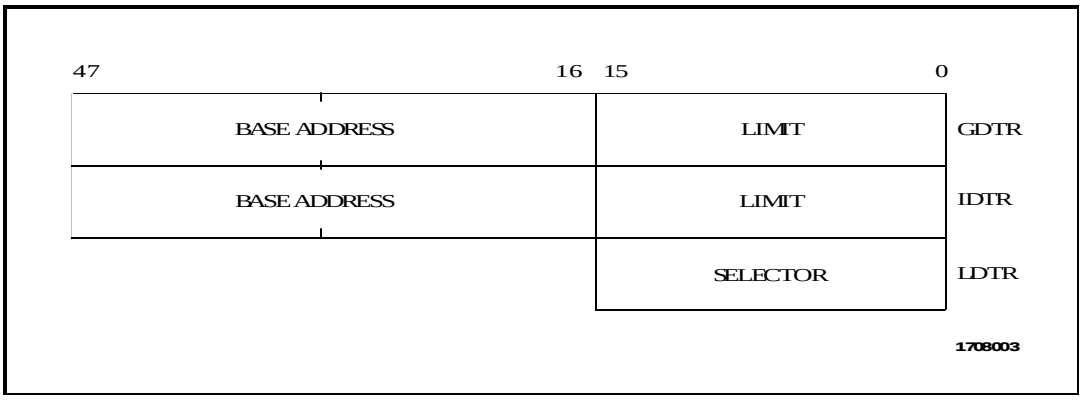


Figure 2-4. Descriptor Table Registers

Subsequent access to entries in the LDT use the hidden LDTR cache to obtain linear addresses. If the LDT descriptor is modified in the GDT, the LDTR must be reloaded to update the hidden portion of the LDTR.

When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the LDT to locate a segment descriptor. If TI = 1, the index portion of the selector is used to locate a given descriptor within the LDT. Each task in the system may be given its own LDT, managed by the operating system. The LDTs provide a method of isolating a given task's segments from other tasks in the system.

The LDTR can be read or written by the LLDT and SLDT instructions.

Descriptors

There are three types of descriptors:

- Application Segment Descriptors that define code, data and stack segments.
- System Segment Descriptors that define an LDT segment or a Task State Segment (TSS) table described later in this text.
- Gate Descriptors that define task gates, interrupt gates, trap gates and call gates.

Application Segment Descriptors can be located in either the LDT or GDT. System Segment Descriptors can only be located in the GDT. Dependent on the gate type, gate descriptors may be located in either the GDT, LDT or IDT. Figure 2-5 illustrates the descriptor format for both Application Segment Descriptors and System Segment Descriptors. Table 2-14 (Page 2-35) lists the corresponding bit definitions.

Table 2-15. (Page 2-35) and Table 2-16. (Page 2-36) defines the DT field within the segment descriptor.

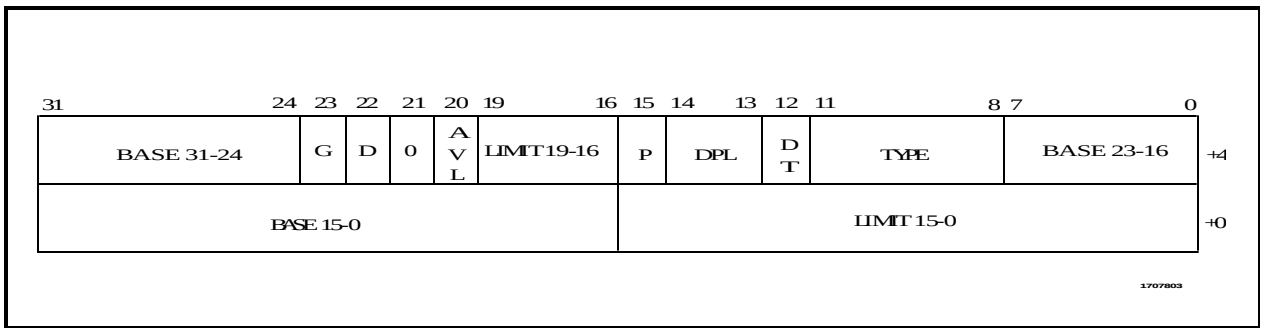


Figure 2-5. Application and System Segment Descriptors

Table 2-14. Segment Descriptor Bit Definitions

BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-24 7-0 31-16	+4 +4 +0	BASE	Segment base address. 32-bit linear address that points to the beginning of the segment.
19-16 15-0	+4 +0	LIMIT	Segment limit.
23	+4	G	Limit granularity bit: 0 = byte granularity, 1 = 4 KB (page) granularity.
22	+4	D	Default length for operands and effective addresses. Valid for code and stack segments only: 0 = 16 bit, 1 = 32-bit.
20	+4	AVL	Segment available.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
12	+4	DT	Descriptor type: 0 = system, 1 = application.
11-8	+4	TYPE	Segment type. See Tables 2-7 and 2-8.

Table 2-15. TYPE Field Definitions with DT = 0

TYPE (BITS 11-8)	DESCRIPTION
0001	TSS-16 descriptor, task not busy.
0010	LDT descriptor.
0011	TSS-16 descriptor, task busy.
1001	TSS-32 descriptor, task not busy
1011	TSS-32 descriptor, task busy.

Table 2-16. TYPE Field Definitions with DT = 1

TYPE				APPLICATION DECRIPTOR INFORMATION
E	C/D	R/W	A	
0	0	x	x	data, expand up, limit is upper bound of segment
0	1	x	x	data, expand down, limit is lower bound of segment
1	0	x	x	executable, non-conforming
1	1	x	x	executable, conforming (runs at privilege level of calling procedure)
0	x	0	x	data, non-writable
0	x	1	x	data, writable
1	x	0	x	executable, non-readable
1	x	1	x	executable, readable
x	x	x	0	not-accessed
x	x	x	1	accessed

Gate Descriptors provide protection for executable segments operating at different privilege levels. Figure 2-7 illustrates the format for Gate Descriptors and the table lists the corresponding bit definitions.

Task Gate Descriptors are used to switch the CPU's context during a task switch. The selector portion of the task gate descriptor locates a Task State Segment. These descriptors can be located in the GDT, LDT or IDT tables.

Interrupt Gate Descriptors are used to enter a hardware interrupt service routine. Trap Gate Descriptors are used to enter exceptions or software interrupt service routines. Trap Gate and Interrupt Gate Descriptors can only be located in the IDT.

Call Gate Descriptors are used to enter a procedure (subroutine) that executes at the same or a more privileged level. A Call Gate Descriptor primarily defines the procedure entry point and the procedure's privilege level.

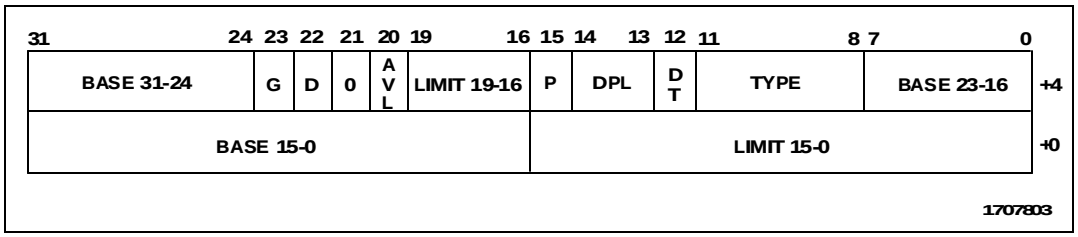


Figure 2-6 Gate Descriptor

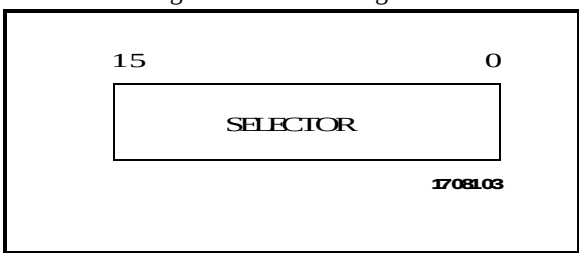
BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-16 15-0	+4 +0	OFFSET	Offset used during a call gate to calculate the branch target.
31-16	+0	SELECTOR	Segment selector used during a call gate to calculate the branch target.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
11-8	+4	TYPE	Segment type: 0100 = 16-bit call gate 0101 = task gate 0110 = 16-bit interrupt gate 0111 = 16-bit trap gate 1100 = 32-bit call gate 1110 = 32-bit interrupt gate 1111 = 32-bit trap gate.
4-0	+4	PARAME-TERS	Number of 32-bit parameters to copy from the caller's stack to the called procedure's stack (valid for calls).

2.4.3 Task Register

The Task Register (TR) holds a 16-bit selector for the current Task State Segment (TSS) table as shown in Figure 2-7. The TR is loaded and stored via the LTR and STR instructions, respectively. The TR can only be accessed during protected mode and can only be loaded when the privilege level is 0 (most privileged). When the TR is loaded, the TR selector field indexes a TSS descriptor that must reside in the Global Descriptor Table (GDT). The contents of the selected descriptor are cached on-chip in the hidden portion of the TR.

During task switching, the processor saves the current CPU state in the TSS before starting a new task. The TR points to the current TSS. The TSS can be either a 386/486-style 32-bit TSS or a 286-style 16-bit TSS type. An I/O permission bit map is referenced in the 32-bit TSS by the I/O Map Base Address.

Figure 2-7. Task Register



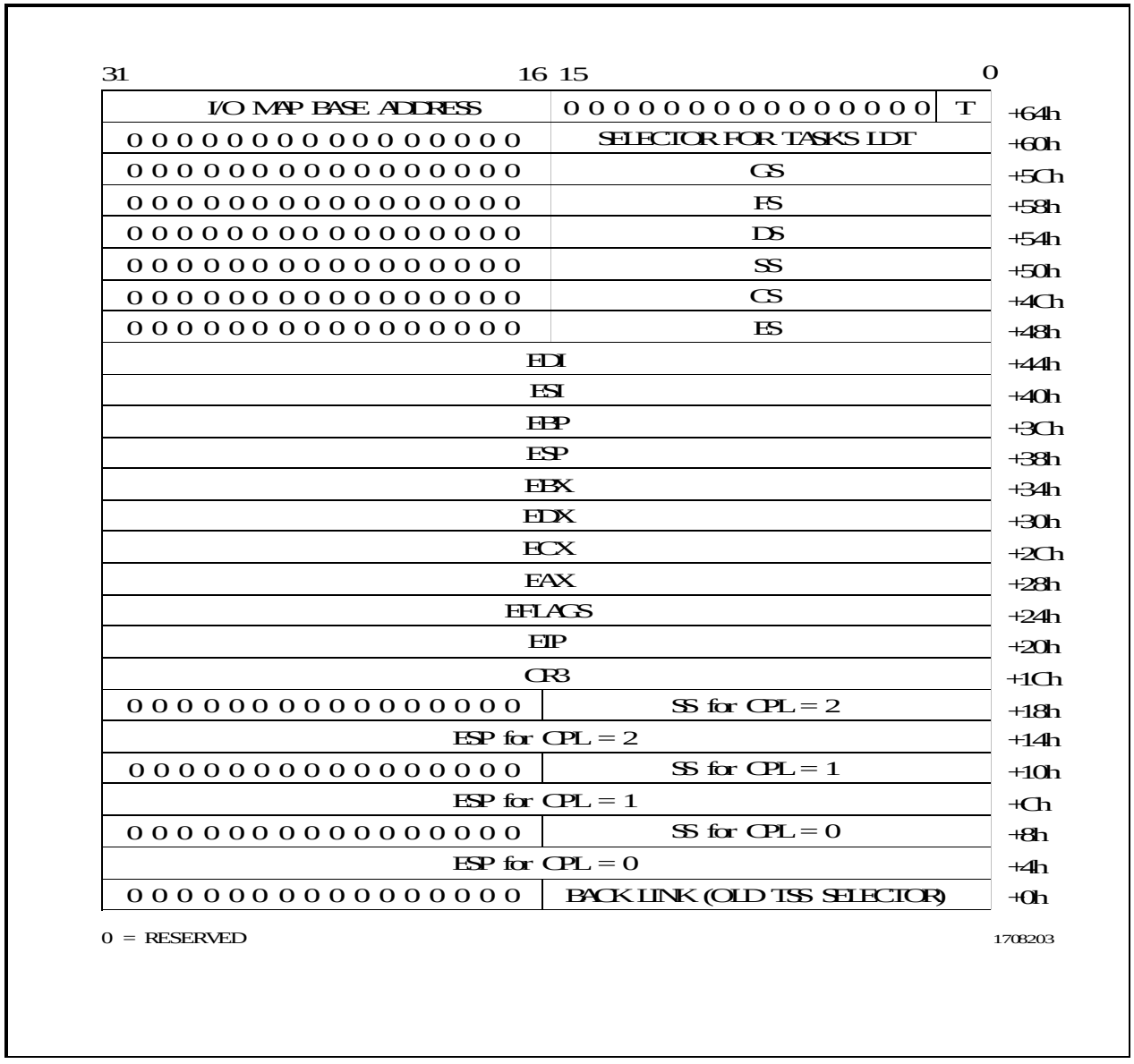


Figure 2-8. 32-Bit Task State Segment (TSS) Table

System Register Set

SELECTOR FOR TASK'S LDT	+2Ah
DS	+28h
SS	+26h
CS	+24h
ES	+22h
DI	+20h
SI	+1Eh
BP	+1Ch
SP	+1Ah
BX	+18h
DX	+16h
CX	+14h
AX	+12h
FLAGS	+10h
IP	+Eh
SS FOR PRIVILEGE LEVEL 2	+Ch
SP FOR PRIVILEGE LEVEL 2	+Ah
SS FOR PRIVILEGE LEVEL 1	+8h
SP FOR PRIVILEGE LEVEL 1	+6h
SS FOR PRIVILEGE LEVEL 0	+4h
SP FOR PRIVILEGE LEVEL 0	+2h
BACK LINK (OLD TSS SELECTOR)	+0h

1708803

Table 2-17. 16-Bit Task State Segment (TSS) Table

2.4.4 Model Specific Registers

The CPU contains several Model Specific Registers (MSR's) that provide time stamp, performance monitoring and counter event functions. Access to a specific MSR through an index value in the ECX register as shown in Table 18 below.

Table 2-18. Model Specific Register

Register Description	ECX Value
Test Data	3h
Test Address	4h
Command/Status	5h
Time Stamp Counter (TSC)	10h
Counter Event Control Register	11h
Performance Counter #0	12h
Performance Counter #1	13h

The MSR registers can be read using the RDMSR instruction, opcode 0F32h. During an MSR register read, the contents of the particular MSR register, specified by the ECX register, is loaded into the EDX:EAX registers.

The MSR registers can be written using the WRMSR instruction, opcode 0F30h. During a MSR register write the contents of EDX:EAX are loaded into the MSR register specified in the ECX register.

The RDMSR and WRMSR instructions are privileged instructions and are also used to setup scratchpad lock.

2.4.4.1 Time Stamp Counter

The Time Stamp Counter (TSC) Register MSR[10] is a 64-bit counter that counts the internal CPU clock cycles since the last reset. The TSC uses a continuous CPU core clock and will continue to count clock cycles even when the processor is in Suspend mode.

The TSC can be accessed using the RDMSR and WRMSR instructions. In addition, the TSC can be read using the RDTSC instruction, opcode 0F31h. The RDTSC instruction loads the contents of the TSC into EDX:EAX. The use of the RDTSC instruction is restricted by the Time Stamp Counter, (TSC) flag in CR4. When the TSC flag is 0, the RDTSC instruction can be executed at any privilege level. When the TSC flag is 1, the RDTSC instruction can only be executed at privilege level0.

2.4.4.2 Performance Monitoring

Performance monitoring allows counting of over a hundred different event occurrences and durations. Two 48-bit counters are used: Performance Monitor Counter 0 and Performance Monitor Counter 1. These two performance monitor counters are controlled by the Counter Event Selection and Control Register MSR[11]. The performance monitor counters use a continuous CPU core clock and will continue to count clock cycles even when the processor is in Suspend or Shutdown mode.

2.4.4.2.1 Performance Monitoring Counters1and2

The 48-bit Performance Monitoring Counter (PMC) Registers MSR[12] and MSR[13] count events as specified by the Counter Event Selection and Control Register, MSR[11].

The PMCs can be accessed by the RDMSR and WRMSR instructions. In addition, the PMCs can be read by the RDPMC instruction, opcode 0F33h. The RDPMC instruction loads the contents of the PMC register specified in the ECX register into EDX:EAX. The use of RDPMC instructions is restricted by the Performance Monitoring Counter Enable, (PCE) flag in CR4.

When the PCE flag is set to 1, the RDPMC instruction can be executed at any privilege level. When the PCE flag is 0, the RDPMC instruction can only be executed at privilege level 0.

2.4.4.2.2 Counter Event Selection and Control Register

Register MSR[11] controls the two internal counters, #0 and #1. The events to be counted have been chosen based on the micro-architecture of the Cyrix III processor. The control register for the two event counters is described in Table 2-19.

2.4.4.2.3 Counter Type Control

The Counter Type bit determines whether the counter will count clocks or events. When counting clocks the counter operates as a timer.

2.4.4.2.4 CPL Control

The Current Privilege Level (CPL) can be used to determine if the counters are enabled. The CP02 bit in the MSR[11] register enables counting when the CPL is less than three, and the CP03 bit enables counting when CPL is equal to three. If both bits are set, counting is not dependent on the CPL level; if neither bit is set, counting is disabled.

Table 2-19. Counter Event Selection and Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD					TC1*	RSVD	CT1	CP13	CP12	TC1*						RSVD					TC0*	RSVD	CT0	CP03	CP02	TC0*					

Note: Split fields.

Table 2-20. Counter Event Selection and Control Register Bit Definitions

Bit	Name	Description
32:27	RSVD	Reserved
25	RSVD	Reserved
24	CT1	**Counter #1 Counter Type:** If = 1: Count clock cycles If = 0: Count events (reset state).
23	CP13	**Counter #1 CPL 3 Enable:** If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (Reset state)
22	CP12	**Counter #1 CPL Less Than 3 Enable:** If = 1: Enable counting when CPL < 3. If = 0: Disable counting when CPL < 3. (Reset state)
26, 21:16	TC1[5:0]	**Counter #1 Event Type:** Reset state = 0
15:9	RSVD	Reserved
8	CT0	**Counter #0 Counter Type**: If = 1: Count clock cycles If = 0: Count events (reset state).
7	CP03	**Counter #0 CPL 3 Enable:** If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (reset state)
6	CP02	**Counter #0 CPL Less Than 3 Enable:** If = 1: Enable counting when CPL < 3. If = 0: Disable counting when CPL < 3. (reset state)
10, 5:0	TC0[5:0]	**Counter #0 Event Type:** Reset state = 0

2.4.5 Event Type and Description

The events that can be counted by the performance monitoring counters are listed in Table 2-21. Each of the 127 event types is assigned an event number.

A particular event number to be counted is placed in one of the MSR[11] Event Type fields. There is a separate field for counter #0 and #1.

The events are divided into two groups. The occurrence type events and duration type events. The occurrence type events, such as hardware interrupts, are counted as single events. The duration type events such as “clock while bus cycles are in progress” count the number of clock cycles that occur during the event.

Table 2-21. Event Type Register

Number	Counter #0	Counter #1	Description	Type
00h	Yes	Yes	Data reads	Occurrence
01h	Yes	Yes	Data writes	Occurrence
02h	Yes	Yes	Data TLB misses	Occurrence
03h	Yes	Yes	Cache misses: Data reads	Occurrence
04h	Yes	Yes	Cache misses: Data writes	Occurrence
05h	Yes	Yes	Data writes that hit on modified or exclusive lines	Occurrence
06h	Yes	Yes	Data cache lines written back	Occurrence
07h	Yes	Yes	External inquiries	Occurrence
08h	Yes	Yes	External inquires that hit	Occurrence
09h	Yes	Yes	Memory accesses in both pipelines	Occurrence
0Ah	Yes	Yes	Cache bank conflicts	Occurrence
0Bh	Yes	Yes	Misaligned data references	Occurrence
0Ch	Yes	Yes	Instruction fetch requests	Occurrence
0Dh	Yes	Yes	L2 TLB code misses	Occurrence
0Eh	Yes	Yes	Cache misses: Instruction fetch	Occurrence
0Fh	Yes	Yes	Any Segment Register load	Occurrence
10h	Yes	Yes	Reserved	Occurrence
11h	Yes	Yes	Reserved	Occurrence
12h	Yes	Yes	Any branch	Occurrence
13h	Yes	Yes	BTB hits	Occurrence
14h	Yes	Yes	Taken branches or BTB hits	Occurrence
15h	Yes	Yes	Pipeline flushes	Occurrence
16h	Yes	Yes	Instructions executed in both pipelines	Occurrence
17h	Yes	Yes	Instructions executed in Y pipeline	Occurrence
18h	Yes	Yes	Clocks while bus cycles are in progress	Duration
19h	Yes	Yes	Pipeline stalled by full write buffers	Duration
1Ah	Yes	Yes	Pipeline stalled by waiting on data memory reads	Duration
1Bh	Yes	Yes	Pipeline stalled by writes to not-modified or not-exclusive cache lines.	Duration

Table 2-21. Event Type Register (Continued)

Number	Counter #0	Counter #1	Description	Type
1Ch	Yes	Yes	Locked bus cycles	Occurrence
1Dh	Yes	Yes	I/O cycles	Occurrence
1Eh	Yes	Yes	Non-cacheable memory requests	Occurrence
1Fh	Yes	Yes	Pipeline stalled by address generation interlock	Duration
20h	Yes	Yes	Reserved	--
21h	Yes	Yes	Reserved	--
22h	Yes	Yes	Floating point operations	Occurrence
23h	Yes	Yes	Breakpoint matches on DR0 register	Occurrence
24h	Yes	Yes	Breakpoint matches on DR1 register	Occurrence
25h	Yes	Yes	Breakpoint matches on DR2 register	Occurrence
26h	Yes	Yes	Breakpoint matches on DR3 register	Occurrence
27h	Yes	Yes	Hardware interrupts	Occurrence
28h	Yes	Yes	Data reads or data writes	Occurrence
29h	Yes	Yes	Data read misses or data write misses	Occurrence
2Bh	Yes	No	MMX instruction executed in X pipeline	Occurrence
2Bh	No	Yes	MMX instruction executed in Y pipeline	Occurrence
2Dh	Yes	No	EMMS instruction executed	Occurrence
2Dh	No	Yes	Transition between MMX instruction and FP instructions	Occurrence
2Eh	No	Yes	Reserved	--
2Fh	Yes	No	Saturating MMX instructions executed	Occurrence
2Fh	No	Yes	Saturations performed	Occurrence
30h	Yes	No	Reserved	--
31h	Yes	No	MMX instruction data reads	Occurrence
32h	Yes	No	Reserved	--
32h	No	Yes	Taken branches	Occurrence
33h	No	Yes	Reserved	--
34h	Yes	No	Reserved	--
34h	No	Yes	Reserved	--
35h	Yes	No	Reserved	--
35h	No	Yes	Reserved	--
36h	Yes	No	Reserved	--
36h	No	Yes	Reserved	--
37h	Yes	No	Returns predicted incorrectly	Occurrence
37h	No	Yes	Return predicted (correctly and incorrectly)	Occurrence
38h	Yes	No	MMX instruction multiply unit interlock	Duration
38h	No	Yes	MODV/MOVQ store stall due to previous operation	Duration
39h	Yes	No	Returns	Occurrence
39h	No	Yes	RSB overflows	Occurrence
3Ah	Yes	No	BTB false entries	Occurrence
3Ah	No	Yes	BTB miss prediction on a not-taken back	Occurrence
3Bh	Yes	No	Number of clock stalled due to full write buffers while executing	Duration

Table 2-21. Event Type Register (Continued)

Number	Counter #0	Counter #1	Description	Type
3Bh	No	Yes	Stall on MMX instruction write to E or M line	Duration
3Ch-3Fh	Yes	Yes	Reserved	--
40h	Yes	Yes	L2 TLB misses (code or data)	Occurrence
41h	Yes	Yes	L1 TLB data miss	Occurrence
42h	Yes	Yes	L1 TLB code miss	Occurrence
43h	Yes	Yes	L1 TLB miss (code or data)	Occurrence
44h	Yes	Yes	TLB flushes	Occurrence
45h	Yes	Yes	TLB page invalidates	Occurrence
46h	Yes	Yes	TLB page invalidates that hit	Occurrence
47h	Yes	Yes	Reserved	--
48h	Yes	Yes	Instructions decoded	Occurrence
49h	Yes	Yes	Reserved	--

2.5 Cyrix III Register Set

The Cyrix III Register Set includes the configuration registers that are used to enable features in the CPU. These features are specific to the Cyrix III processor architecture and are typically only accessed during the boot/initialization process.

Registers included in this section are:

- Configuration Control Registers
- Address Region Registers
- Region Control Registers

2.5.1 I/O Port 22h and 23h Access

Access to internal registers is accomplished via an 8-bit index/data I/O pair. Each data transfer, I/O Port 23h, must be preceded by a valid index write, I/O Port 22h. All reads from I/O Port 22h produce external I/O cycles; therefore, the index cannot be read. Accesses that hit within the on-chip configuration registers do not generate external I/O cycles.

To access the above registers, the programmer uses the Port 22h (index) and Port 23h (data). The access is atomic when an SMI is involved, but is not atomic when any of the following three conditions are presented: 1) INT, 2) NMI 3) INIT#. Proper steps must be taken to inhibit these three conditions if a pure atomic operation is to be achieved. An example of this to prevent an INT sequence would be to use a PUSHF, CLI instruction pair before doing the Port 22h/23h access with a POPF after the access has been done.

After reset, only configuration registers with indices C0-CFh and FC-FFh are accessible. This prevents potential conflicts with other devices that use Ports 22h and 23h to access their registers.

2.5.2 Map Enable (MAPEN)

The purpose of MAPEN is to increase the number of registers that can be accessed via Ports 22h/23h. The MAPEN fields can be thought of as a paging mechanism to the complete register set allowing multiple registers to share the same index value but providing different functionality. MAPEN must be set correctly to gain access to the register that is to be modified. MAPEN is located in CCR3[7:4]. It is strongly recommended that the programmer use the following sequence:

1) Read CCR3.
2) Save CCR3.
3) Modify MAPEN at CCR3[7-4].
4) Access Register via Port 22h/23h access.
5) Restore CCR3 to control the MAPEN field.

There are 256 possible registers available for each MAPEN setting, for a total of 4096. Most registers are not defined and are therefore reserved.

2.5.3 Cyrix Configuration Control Registers

Table 2-22 summarizes the Core Registers. The registers are described in greater detail beginning on page 52.

Table 2-22. CPU Configuration Register Summary

Acronym	Register	Index	Size (Bits)	MAPEN CCR3[7-4]	Reference
CCR0	Configuration Control 0	C0h	8	xxxxb	
CCR1	Configuration Control 1	C1h			
CCR2	Configuration Control 2	C2h			
CCR3	Configuration Control 3	C3h			
CCR4	Configuration Control 4	E8h		0001b	
CCR5	Configuration Control 5	E9h			
CCR6	Configuration Control 6	EAh			
CCR7	Configuration Control 7	EBh			
ARR0	Address Region 0	C4h-C6h	24	xxxxb	
ARR1	Address Region 1	C7h-C9h			
ARR2	Address Region 2	CAh-CCh			
ARR3	Address Region 3	CDh-CFh			
ARR4	Address Region 4	Dh0-D2h		0001b	
ARR5	Address Region 5	D3h-D5h			
ARR6	Address Region 6	D6h-D8h			
ARR7	Address Region 7	D9h-DBh			
ARR8	Address Region 8	A4h-A6h		read= x010b write=x01xb	
ARR9	Address Region 9	A7h-A9h			
ARRA	Address Region A	AAh-ACh			
ARRB	Address Region B	ADh-AFh			
ARRC	Address Region C	D0h-D2h			
ARRD	Address Region D	D3h-D5h			
RCR0	Region Configuration Register 0	DCh	8	0001b	
RCR1	Region Configuration Register 1	DDh			

Table 2-22. CPU Configuration Register Summary (Continued)

Acronym	Register	Index	Size (Bits)	MAPEN CCR3[7-4]	Reference
RCR2	Region Configuration Register 2	DEh	8	x00xb	
RCR3	Region Configuration Register 3	DFh			
RCR4	Region Configuration Register 4	E0h			
RCR5	Region Configuration Register 5	E1h			
RCR6	Region Configuration Register 6	E2h			
RCR7	Region Configuration Register 7	E3h			
RCR8	Region Configuration Register 8	DCh			
RCR9	Region Configuration Register 9	DDh			
RCRA	Region Configuration Register A	DEh			
RCRB	Region Configuration Register B	DFh			
RCRC	Region Configuration Register C	E0h			
RCRD	Region Configuration Register D	E1h			
DIR0	Directory Register 0	FEh		xxxxb	
DIR1	Directory Register 1	FFh			
DIR2	Directory Register 2	FDh			
DIR3	Directory Register 3	FCh		read= x010b write=x01xb	
DIR4	Directory Register 4	FBh			
BCR1	BIOS Core to Bus Clock Ratio	48h		0100b	
BCR1	BIOS PLL Hot Reset	49h			
LCR1	L2_CNTL	41h			
TWR0	Table Walk 0	20h		0001b	

Note: Registers and MAPEN values not mentioned in this section are reserved.

2.5.4 Cyrix Configuration Control Registers (CCR0-CCR7)

The Configuration Control Registers (CCR0-CCR7) are used to assign non-cached memory areas, set up SMM, provide CPU identification information and control various features.

CCR1, CCR3, and CCR6 may be written at any time unless the SMI_LOCK (CCR3[0]) is set or an SMI is active.

2.5.4.1 Configuration Control Register 0 (CCR0)

7	6	5	4	3	2	1	0
Reserved						NC1	*Reserved*

Index: C0h
Default Value: 02h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
1	NC1	No Cache 640 KB - 1 MB 1 = Address region 640 KB to 1 MB is non-cacheable. 0 = Address region 640 KB to 1 MB is cacheable.

2.5.4.2 Configuration Control Register 1 (CCR1)

7	6	5	4	3	2	1	0
SM3	*Reserved*						

Index: C1h
Default Value: 20h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7	SM3	SMM Address Space Address Region 3: 1 = Address Region 3 is designated as SMM address space. 0 = Address Region 3 is system memory.
1	Reserved	Read only (USE_SMI). Set to 1.

This register may be written at any time unless the SMI_LOCK (CCR3[0]) is set or an SMI is active.

2.5.4.3 Configuration Control Register 2 (CCR2)

7	6	5	4	3	2	1	0
Reserved			WPR1	SUSP_HLT	LOCK_NW	*Reserved*	

Index: C2h
Default Value: 00h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
4	WPR1	Write-Protect Region 1 1 = Designates any cacheable accesses in 640 KB to 1 MB address region are write protected.
3	SUSP_HLT	Suspend on Halt: 1 = Execution of the HLT instruction causes the CPU to enter low power suspend mode. 0 = Halt behaves normally.
2	LOCK_NW	Lock NW: 1 = NW bit (CR0[29]) becomes read-only and the CPU ignores any writes to the NW bit. 0 = NW bit (CR0[29]) can be modified.

2.5.4.4 Configuration Control Register 3 (CCR3)

7	6	5	4	3	2	1	0
MAPEN[3-0]				*Reserved*		NMI_EN	SMI_LOCK

Index: C3h
Default Value: 00h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7:4	MAPEN[3-0]	**Map Enable Bits** **These four bits enable different combinations of configuration registers.** Refer to Section 2.5.2 "Map Enable (MAPEN)" on page 47. 0001 = All configuration registers are accessible. 0000 = Only configuration register with indexes: C0 - CFh, FEh, and FFh are accessible
1	NMI_EN	NMI Enable: 1 = NMI interrupt is recognized while servicing an SMI interrupt. NMI_EN should be set only while in SMM after the appropriate SMI interrupt service routine has been set up. 0 = NMI disabled while servicing SMI.
0	SMI_LOCK	SMI Lock: 1 = The following SMM configuration bits can only be modified while in an SMI service routine: CCR1: USE_SMI, SMAC, SM3 CCR3: NMI_EN CCR6: N, SMM_MODE ARR3: Starting address and block size. Once set, the features locked by SMI_LOCK cannot be unlocked until the RESET pin is asserted.

This register contains MAPEN which is a pointer to the group of registers. See MAPEN Section 2.5.2 on page 2-47.

This register may be written at any time unless the SMI_LOCK is set or an SMI is active.

2.5.4.5 Configuration Control Register 4 (CCR4)

7	6	5	4	3	2	1	0
CPUID	Reserved						

Index: E8h
Default Value: 84h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
7	CPUID	Enable CPUID Instructions: 1 = The ID bit in the FFLAGS register can be modified and execution of the CPUID instruction occurs. 0 = The ID bit in the EFLAGS register can not be modified and execution of the CPUID instruction causes an invalid opcode exception.

2.5.4.6 Configuration Control Register 5 (CCR5)

7	6	5	4	3	2	1	0
Reserved		ARREN	*Reserved*				

Index: E9h
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
5	ARREN	**Address Region Registers Enable:** Enables address decoding of ARR0-ARRD. 1 = Enables all ARR registers. 0 = Disables all ARR registers. If SM3 is set ARR3 is enabled regardless of the setting of ARREN. (SM3 is bit 7 in CCR1.)

2.5.4.7 Configuration Control Register 6 (CCR6)

7	6	5	4	3	2	1	0
Reserved							SMM_MODE

Index: EAh:
Default Value: 40h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
0	SMM_MODE	SMM Mode: 1 = Enable Cyrix-enhanced SMM mode. 0 = Disable Cyrix-enhanced SMM mode.

This register may be written at any time unless the SMI_LOCK (CCR3[0]) is set or the processor is in SMM.

2.5.4.8 Configuration Control Register 7 (CCR7)

7	6	5	4	3	2	1	0
Reserved	*Reserved*	*Reserved*	3DNOW_EN	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Index: EBh
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
4	3DNOW_EN	1 = Enable 3DNOW instructions. 0 = Disable 3DNOW instructions.

2.5.4.9 Table Walk Register 0 (TWR0)

7	6	5	4	3	2	1	0
Reserved	*Reserved*	*Reserved*	Cache_TE	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Index: 20h
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
4	Cache_TE	1 = Enable caching of table entries. Improves performance. 0 = Disable caching of table entries.

2.5.4.10 Address Regions in Memory

Selected regions of main memory space can be assigned different attributes. These regions are called address regions. Each address region is defined by a pair of registers—an Address Region Register (ARRn) and a Region Control Register (RCRn).

The ARRn registers are used to specify the location and size for these regions.

The RCRn registers are used to specify the attributes for these regions.

The number (n) is a hexadecimal number that designates the region number.

2.5.4.11 Address Region Registers (ARRn)

23 16	15 8	7 4	3 0
MAIN MEMORY BASE ADDRESS			SIZE

Index: See Table 2-23
Default Value: 00h
Access: Read/Write
MAPEN: See next page.

Bit	Name	Description
23-4	MAIN MEMORY BASE ADDRESS	Starting address for the particular address region. Memory address bits A[31-24] defined by ARRn bits 23 - 16 Memory address bits A[23-16] defined by ARRn bits 15 - 8 Memory address bits A[15-12] defined by ARRn bits 7 - 4
3-0	SIZE	Size of the particular address region as defined by Table 2-24 (Page 2-62).

Three I/O 22h/23h port passes are required access one of 24-bit ARRn registers. The three index numbers for all the ARRn registers are listed in Table 2-23.

Table 2-23. ARRn Register Index Assignment

ARRn Name	MAIN MEMORY BASE ADDRESS			SIZE	MAPEN CCR3[7-4]
	A31-A24	A23-A16	A15-A12		
ARR0	C4h	C5h	C6h[7-4]	C6h[3-0]	xxxxb
ARR1	C7h	C8h	C9h[7-4]	C9h[3-0]	
ARR2	CAh	CBh	CCh[7-4]	CCh[3-0]	
ARR3	CDh	CEh	CFh[7-4]	CFh[3-0]	
ARR4	D0h	D1h	D2h[7-4]	D2h[3-0]	0001b
ARR5	D3h	D4h	D5h[7-4]	D5h[3-0]	
ARR6	D6h	D7h	D8h[7-4]	D4h[3-0]	
ARR7	D9h	DAh	DBh[7-4]	DBh[3-0]	
ARR8	A4h	A5h	A6h[7-4]	A6h[3-0]	read= x010b write=x01xb
ARR9	A7h	A8h	A9h[7-4]	A9h[3-0]	
ARRA	AAh	ABh	ACh[7-4]	ACh[3-0]	
ARRB	ADh	AEh	AFh[7-4]	AFh[3-0]	
ARRC	D0h	D1h	D2h[7-4]	D2h[3-0]	
ARRD	D3h	D4h	D5h[7-4]	D5h[3-0]	

Address region 7 defines total system memory unless superseded by other address region definitions. The SIZE field is defined in two different way as listed in Table 2-24. If the address region size field is zero, the address region is disabled. After a reset, all ARR Registers are initialized to 00h. The base address of the ARR address region, selected by the Base Address field, must be on a block size boundary. For example, a 128KB block is allowed to have a starting address of 0KB, 128KB, 256KB, and so on. A 512KB block is allowed to have a starting address of 0KB, 512KB, 1024KB, and so on. Address region 3 defines SMM space when enabled by CCR1 bit 7.

Table 2-24. ARRn Address Region Size Field Definitions

Size	Block Size		
	ARR0-ARR6	ARR7-ARRB	ARRC-ARRD
00h	Disable	Disable	Disable
01h	4 KB	256 KB	256KB
02h	8 KB	512 KB	Reserved
03h	16 KB	1 MB	
04h	32 KB	2 MB	
05h	64 KB	4 MB	
06h	128 KB	8 MB	
07h	256 KB	16 MB	
08h	512 KB	32 MB	
09h	1 MB	64 MB	
0Ah	2 MB	128 MB	
0Bh	4 MB	256 MB	
0Ch	8 MB	512 MB	
0Dh	16 MB	1 GB	
0Eh	32 MB	2 GB	
0Fh	4 GB	4 GB	

2.5.4.12 Region Control Registers

7	6	5	4	3	2	1	0
Reserved	INV_RGN	WP	WT	WG	*Reserved*		RCD/RDE

*Note: RCD is defined for RCR0-RCR6. RCE is defined for RCR7.

BIT POSITION	NAME	DESCRIPTION
6	INV_RGN	ARR0 Invert Region 1 = applies the controls specified in RCRn to all memory addresses outside the region specified in ARR0.
5	WP	Write-Protect 1 = enables write protect for address region n.
4	WT	Write-Through 1 = defines the address region as write through instead of write-back. Any system ROM that is allowed to be cached by the processor should be defined as write through.
3	WG	Write-Gathering 1 = enables write gathering for address region n. With WG enabled, multiple byte, word or dword writes to sequential addresses that would normally occur as individual cycles on the bus are collapsed, or “gathered” within the processor and then completed as a single write cycle. WG improves bus utilization and should be used on memory regions that are not sensitive to gathering.
0	RCD	Cache Disable (RCR0 - RCR6 only) 1 = defines the address region n as non-cacheable.
0	RCE	Cache Enable (RCR7 only) 1 = enables caching of all memory outside of con-cachable regions

The Region Control Registers (RCRn) specify the attributes for the memory address regions defined by the corresponding Address Region Registers (ARRn). Cacheability, weak locking, write gathering and cache write-through policies can be enabled or disabled using the RCRn registers. Table 2-25 describes the index and MAPEN assignments for RCRn.

Undefined Memory Regions

If an address is accessed that is not in a memory region defined by an ARR/RCR register pair, the following conditions apply:

- If the memory address is cached, write-back is enabled
- Writes are not gathered

Overlapping Regions

If two regions specified by ARR Registers overlap and conflicting attributes are specified, the following attributes take precedence:

- Write-back is disabled
- Writes are not gathered
- Strong locking takes place
- The overlapping regions are non-cacheable

Table 2-25. RCRn Register Index Assignment

RCRn Name	Index	MAPEN CCR3[7-4]
RCR0	DCh	x00xb
RCR1	DDh	
RCR2	DEh	
RCR3	DFh	
RCR4	E0h	
RCR5	E1h	
RCR6	E2h	
RCR7	E3h	
RCR8	DCh	read= x010b write=x01xb
RCR9	DDh	
RCRA	DEh	
RCRB	DFh	
RCRC	E0h	
RCRD	E1h	

Inverted Region (INV_RGN)

Setting INV_RGN applies the controls in RCRx to all the memory addresses outside the specified address region ARRx. This bit affects RCR0-RCR6, but not RCR7

Write-Through (WT)

Setting WT defines the address region as write-through instead of write-back, assuming the region is cacheable. Regions where system ROM are loaded (shadowed or not) should be defined as write-through.

Write Gathering (WG)

Setting WG enables write gathering for the associated address region. Write gathering allows multiple byte, word, or DWORD sequential address writes to accumulate in the on-chip write buffer. As instructions are exe-

cuted the results are placed in a series of output buffers. These buffers are gathered into the final output buffer.

- When the 32-byte buffer becomes full, the contents of the buffer are written on the external 64-bit data bus. Performance is enhanced by avoiding many memory write cycles.
- WG should not be used on memory regions that are sensitive to write cycle gathering. WG can be enabled for both cacheable and non-cacheable regions.

Write Protect (WP)

- Setting WP enables write protect for the corresponding address region. With WP enabled, The memory region is treated as read-only when cached into the cpu cache. During a cache-hit write, the cache will not be modified. The data will still be written through to main memory however, and it is up to the chipset memory controller to ignore the main memory write if necessary.

Cache Disable (CD)

- Cache Disable, if set, defines the address region as non-cacheable. This bit works in conjunction with the *CR0_CD and PCD bits* to determine line cacheability. Whenever possible, the ARR/RCR combination should be used to define non-cacheable regions.

For RCR0 through RCR6

- Only one RCD, WG, WT. WP bit of each RCR register can be SET at a time.
- To define a region to be WC, define corresponding ARR6-0 and SET only WG bit in the corresponding RCR6-0 to 1.
- To define a region to be WP, define corresponding RCR6-0 and SET only WP bit in the corresponding RCR6-0 to 1.

ARR/RCR Programming Example

The following example illustrates the values used to program ARR/RCR registers. In this example, ARR4 is available. Index D0h is the most significant byte of this register; Index D2h is the least significant byte of this register. Values are programmed via the Port 22h/23h mechanism.

Scenario:

- Define a region from 31MB-32MB of memory as non-cacheable and not located in physical memory space:

1) ARR4 (MAPEN = 0001, Index D0h)= 01h	Sets AD31-AD24 to 01h
2) ARR4 (MAPEN = 0001, Index D1h)= F0h	Sets AD23-AD16 to F0h
3) ARR4 (MAPEN = 0001, Index D2h)= 09h	Sets AD15-AD12 to 0h and Size = 1MB (01F0000h = 31M)
4) RCR4 (MAPEN = 0001, Index E0h= 01h	Sets Memory Region to Non-Cacheable

2.5.5 Directory Registers

The DIR Registers allow BIOS and other software to identify the specific CPU and stepping. System Management Mode (SMM) control information is stored in the SMM registers. DIR0 and DIR1 registers are accessed by writing to the I/O Port 22h index using indices FEh and FFh. They can be read from any MAPEN except 4.

2.5.5.1 Directory Register 0 (DIR0)

7	6	5	4	3	2	1	0
Cyrix Processor Family = 8h				CPU Clock Multiplier			

Index: FEh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN

Bit	Name	Description
7-4	Cyrix Processor Family	Cyrix Processor family code. Read only value set to 8.
3-0	CLK_MULT	Clock Multiplier Indicates the clock ratio. The value of CLK_MULT is set by the input pins NMI, INTR, A20M#, IGNNE#; or by BIOS through BCR1 and BCR2

TYPE Bits	Clock Ratio
4h	2.5
1h	3.0
5h	3.5
2h	4.0
6h	4.5
3h	5.0
7h	5.5
8h	6.0
Ah	6.5
9h	7.0
Bh	7.5

2.5.5.2 Directory Register 1 (DIR1)

7	6	5	4	3	2	1	0
STEP_ID				REV_ID			

Index: FFh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: xxxh

Bit	Name	Description
7-4	STEP_ID	Step Identification Indicates the major revision of the Cyrix III. This value is zero based and is usually only incremented on production revision parts.
3-0	REV_ID	Revision Identification Indicates the minor revision of the Cyrix III. This value is zero based. For example, the production revision 1.1 of a CPU is listed as 01h. This value is incremented on production revision parts.

DIR2 is reserved.

2.5.5.3 Directory Register 3 (DIR3)

7	6	5	4	3	2	1	0
TYPE				FAMILY			

Index: FCh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: Read =xxxxb, Write = x00xb

Bit	Name	Description
7 - 4	TYPE	Processor type as read by CPU_ID instruction. Read only value set to 0.
3 - 0	FAMILY	Processor family as read by CPU_ID instruction. Read only value set to 6.

2.5.5.4 Directory Register 4 (DIR0)

7	6	5	4	3	2	1	0
STEP_ID				REV_ID			

Index: FBh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: Read =xxxxb, Write = x00xb

Bit	Name	Description
7-4	MODEL	Processor model as read by CPU_ID instruction. Read only value set to 5.
3-0	STEPPING	Processor stepping as read by CPU_ID instruction.

2.5.5.5 BIOS Core-to-Bus Clock Ratio Configuration Register

Index: 48h
Default Value: 00h
Access: Read/Write
MAPEN: 0100b

Bit	Name	Description
7	HOTRST_TRIGGERED	A read/write bit used to indicate to the BIOS whether hot reset has been triggered or not.
6	Reserved.	0
5:4	BSEL[1- 0]	Indicate the P6 bus speed.
3:0	BIOS_CLKRATIO[3-0]	Core-to-bus clock ratio as described below.

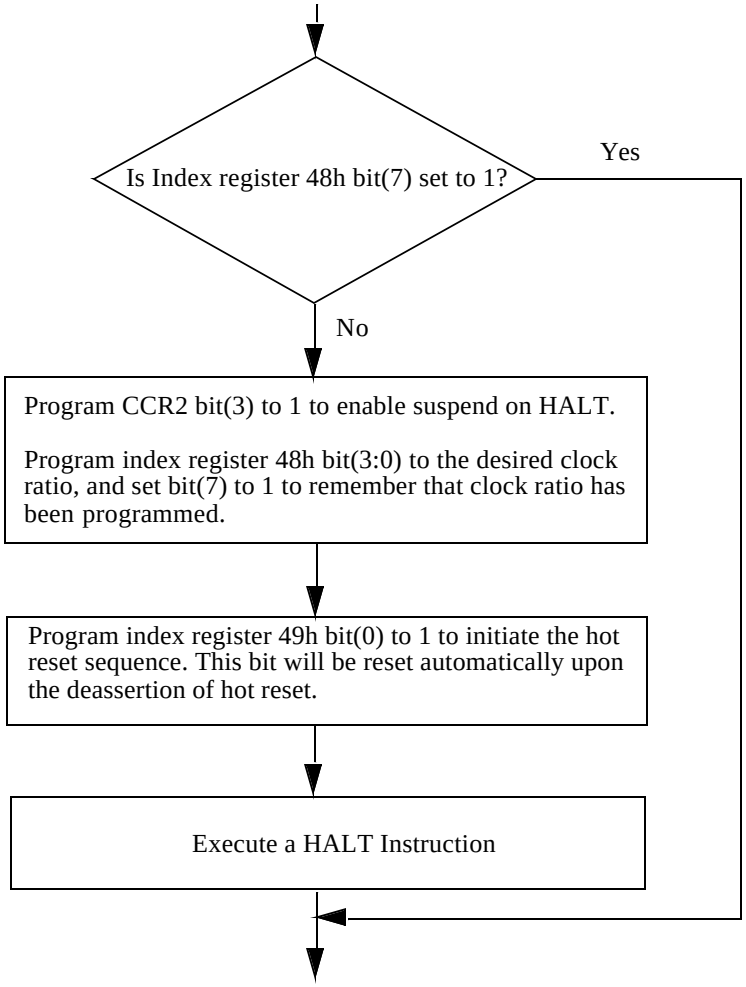
TYPE Bits	Clock Ratio
4h	2.5
1h	3.0
5h	3.5
2h	4.0
6h	4.5
3h	5.0
7h	5.5
8h	6.0
Ah	6.5
9h	7.0
Bh	7.5

Steps to set clock multiplier through BIOS

2.5.5.6 BIOS PLL Hot Reset Configuration Register

Index: 49h
Default Value: 00h
Access: Read/Write
MAPEN: 010

Bit	Name	Description
7:1	Reserved.	0
0	BIOS_HOTRESET	Writing a 1 to this bit will start the internal reset sequence to the PLL and load the BIOS_CLKRATIO[3:0] value to the PLL.



2.5.5.7 L2_CNTL Configuration Register

Index: 41h
Default Value:
Access: Read/Write
MAPEN: 0100b

Bit	Name	Description
3	L2_WT	L2 Write Through. All L1 evictions (cache line writes) are not stored in the L2. If the evicted cache line is modified, the cache line is written to the P6 bus. If the cache line is in the shared or exclusive state, it is discarded.
2	L2_ENABLE	L2 Enable. All L2 accesses (reads, writes or snoops) will miss the L2. An eviction from the L1 (cache line write) will not update the L2. A WBINV instruction must be generated when changing this bit from a 1 to a 0. The POR value of this bit is 0.

2.6 Debug Registers

Six debug registers (DR0-DR3, DR6 and DR7), shown in Figure 2-9, support debugging on the Cyrix III CPU. The bit definitions for the debug registers are listed in Table 2-26 (Page 2-74).

Memory addresses loaded in the debug registers, referred to as “breakpoints”, generate a debug exception when a memory access of the specified type occurs to the specified address. A data breakpoint can be specified for a particular kind of memory access such as a read or a write. Code breakpoints can also be set allowing debug exceptions to occur whenever a given code access (execution) occurs.

The size of the debug target can be set to 1, 2, or 4bytes. The debug registers are accessed via MOV instructions which can be executed only at privilege level 0.

The Debug Address Registers (DR0-DR3) each contain the linear address for one of four possible breakpoints. Each breakpoint is further specified by bits in the Debug Control Register (DR7). For each breakpoint address in DR0-DR3, there are corresponding fields L, R/W, and LEN in DR7 that specify the type of memory access associated with the breakpoint.

The R/W field can be used to specify instruction execution as well as data access breakpoints. Instruction execution breakpoints are always taken before execution of the instruction that matches the breakpoint.

The Debug Status Register (DR6) reflects conditions that were in effect at the time the debug exception occurred. The contents of the DR6 register are not automatically cleared by the processor after a debug exception occurs and, therefore, should be cleared by software at the appropriate time.

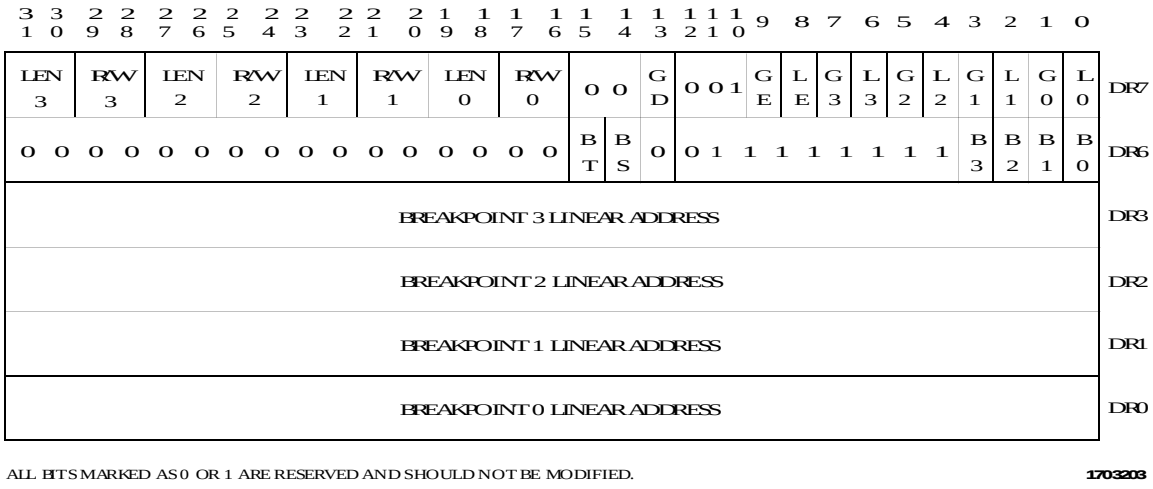


Figure 2-9. Debug Registers

Code execution breakpoints may also be generated by placing the breakpoint instruction (INT 3) at the location where control is to be regained. Additionally, the single-step feature may be enabled by setting the TF flag in the EFLAGS register. This causes the processor to perform a debug exception after the execution of every instruction.

Table 2-26. DR6 and DR7 Debug Register Field Definitions

REGISTER	FIELD	NUMBER OF BITS	DESCRIPTION
DR6	BI	1	BI is set by the processor if the conditions described by DRI, R/Wi, and LENI occurred when the debug exception occurred, even if the breakpoint is not enabled via the GI or LI bits.
	BT	1	BT is set by the processor before entering the debug handler if a task switch has occurred to a task with the T bit in the TSS set.
	BS	1	BS is set by the processor if the debug exception was triggered by the single-step execution mode (TF flag in EFLAGS set).
DR7	R/Wi	2	Specifies type of break for the linear address in DR0, DR1, DR3, DR4: 00 - Break on instruction execution only 01 - Break on data writes only 10 - Not used 11 - Break on data reads or writes.
	LENI	2	Specifies length of the linear address in DR0, DR1, DR3, DR4: 00 - One byte length 01 - Two byte length 10 - Not used 11 - Four byte length.
	Gi	1	If set to a 1, breakpoint in DRI is globally enabled for all tasks and is not cleared by the processor as the result of a task switch.
	LI	1	If set to a 1, breakpoint in DRI is locally enabled for the current task and is cleared by the processor as the result of a task switch.
	GD	1	Global disable of debug register access. GD bit is cleared whenever a debug exception occurs.

2.7 Address Space

The Cyrix III CPU can directly address 64 KB of I/O space and 4 GB of physical memory (Figure 2-24).

Memory Address Space. Access can be made to memory addresses between 0000 0000h and FFFF FFFFh. This 4 GB memory space can be accessed using byte, word (16 bits), or doubleword (32 bits) format. Words and doublewords are stored in consecutive memory bytes with the low-order byte located in the lowest address. The physical address of a word or doubleword is the byte address of the low-order byte.

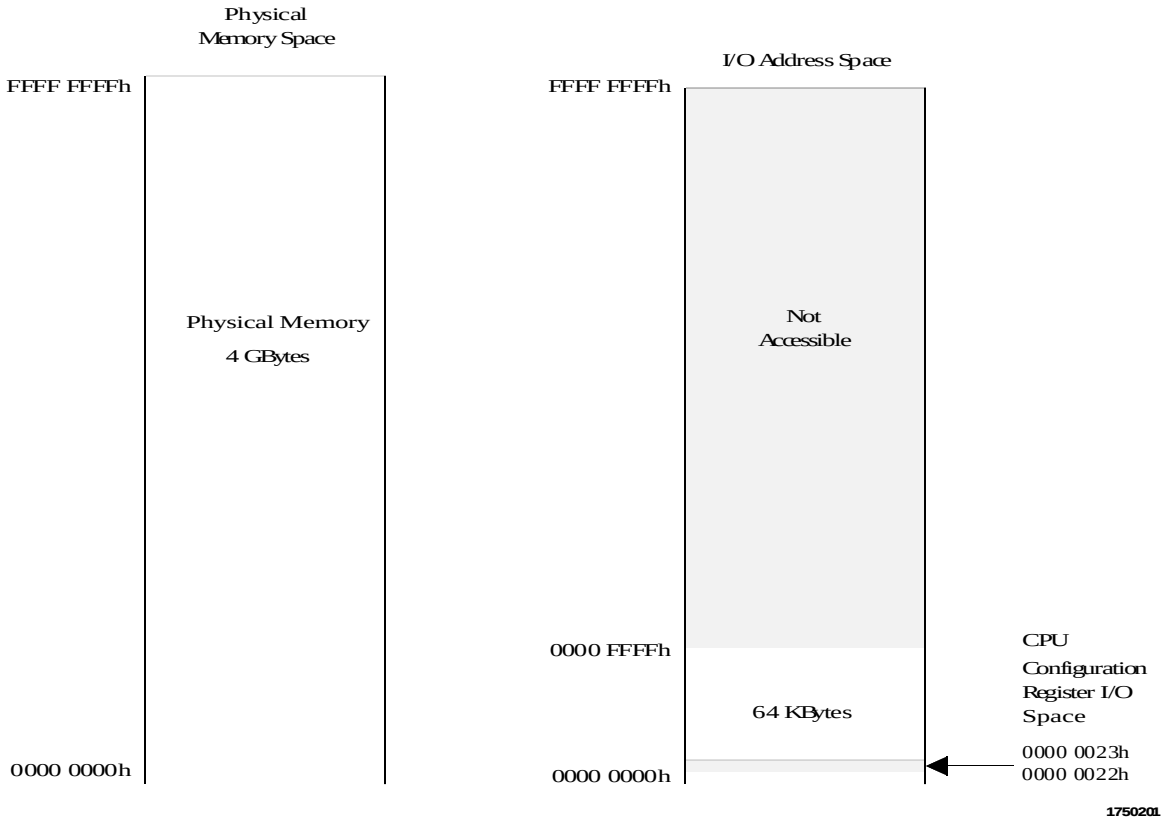


Figure 2-24. Memory and I/O Address Spaces

I/O Address Space

The Cyrix III I/O address space is accessed using IN and OUT instructions to addresses referred to as “ports”. The accessible I/O address space size is 64 KB and can be accessed through 8-bit, 16-bit or 32-bit ports.

The accessible I/O address space ranges between locations 0000 0000h and 0000FFFFh (64 KB). The I/O locations (ports) 22h and 23h can be used to access the Cyrix III configuration registers.

2.8 Memory Addressing Methods

With the Cyrix III CPU, memory can be addressed using nine different addressing modes (Table 2-26, Page 2-77). These addressing modes are used to calculate an offset address often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined using memory management mechanisms to create a physical address that actually addresses the physical memory devices.

Memory management mechanisms on the Cyrix III CPU consist of segmentation and paging. Segmentation allows each program to use several independent, protected address spaces. Paging supports a memory subsystem that simulates a large address space using a small amount of RAM and disk storage for physical memory. Either or both of these mechanisms can be used for management of the Cyrix III CPU memory address space.

2.8.1 Offset Mechanism

The offset mechanism computes an offset (effective) address by adding together one or more of three values: a base, an index and a displacement. When present, the base is the value of one of the eight 32-bit general registers. The index if present, like the base, is a value that is in one of the eight 32-bit general purpose registers (not including the ESP register). The index differs from the base in that the index is first multiplied by a scale factor of 1, 2, 4 or 8 before the summation is made. The third component added to the memory address calculation is the displacement. The displacement is a value of up to 32-bits in length supplied as part of the instruction. Figure 2-25 illustrates the calculation of the offset address.

Nine valid combinations of the base, index, scale factor and displacement can be used with the Cyrix III CPU instruction set. These combinations are listed in Table 2-26. The base and index both refer to contents of a register as indicated by [Base] and [Index].

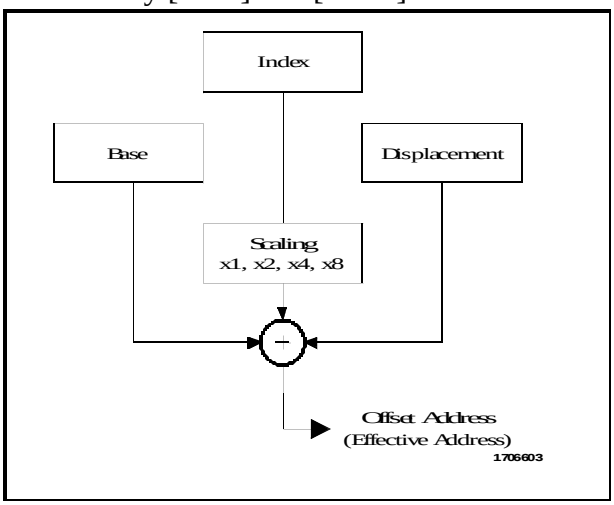


Figure 2-25. Offset Address Calculation

Table 2-26. Memory Addressing Modes

ADDRESSING MODE	BASE	INDEX	SCALE FACTOR (SF)	DISPLACEMENT (DP)	OFFSET ADDRESS (OA) CALCULATION
Direct				x	OA = DP
Register Indirect	x				OA = [BASE]
Based	x			x	OA = [BASE] + DP
Index		x		x	OA = [INDEX] + DP
Scaled Index		x	x	x	OA = ([INDEX] * SF) + DP
Based Index	x	x			OA = [BASE] + [INDEX]
Based Scaled Index	x	x	x		OA = [BASE] + ([INDEX] * SF)
Based Index with Displacement	x	x		x	OA = [BASE] + [INDEX] + DP
Based Scaled Index with Displacement	x	x	x	x	OA = [BASE] + ([INDEX] * SF) + DP

2.8.2 Memory Addressing

Real Mode Memory Addressing

In real mode operation, the Cyrix III CPU only addresses the lowest 1 MB of memory. To calculate a physical memory address, the 16-bit segment base address located in the selected segment register is multiplied by 16 and then the 16-bit offset address is added. The resulting 20-bit address is then extended. Three hexadecimal zeros are added as upper address bits to create the 32-bit physical address. Figure 2-26 illustrates the real mode address calculation.

The addition of the base address and the offset address may result in a carry. Therefore, the resulting address may actually contain up to 21 significant address bits that can address memory in the first 64 KB above 1 MB.

Protected Mode Memory Addressing

In protected mode three mechanisms calculate a physical memory address (Figure 2-27, Page 2-79).

- Offset Mechanism that produces the offset or effective address as in real mode.
- Selector Mechanism that produces the base address.
- Optional Paging Mechanism that translates a linear address to the physical memory address.

The offset and base address are added together to produce the linear address. If paging is not enabled, the linear address is used as the physical memory address. If paging is enabled, the paging mechanism is used to translate the linear address into the physical address. The offset mechanism is described earlier in this section and applies to both real and protected mode. The selector and paging mechanisms are described in the following paragraphs.

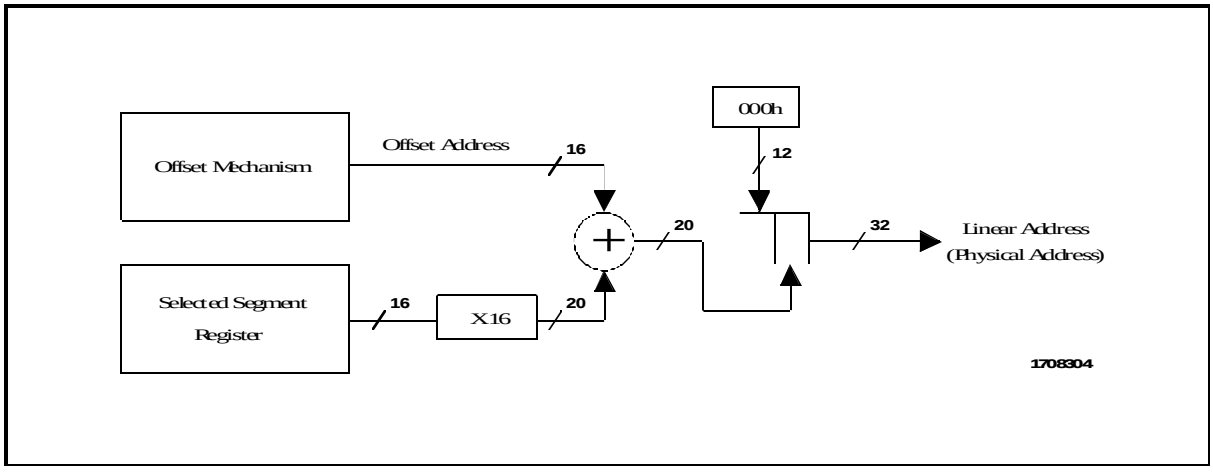


Figure 2-26. Real Mode Address Calculation

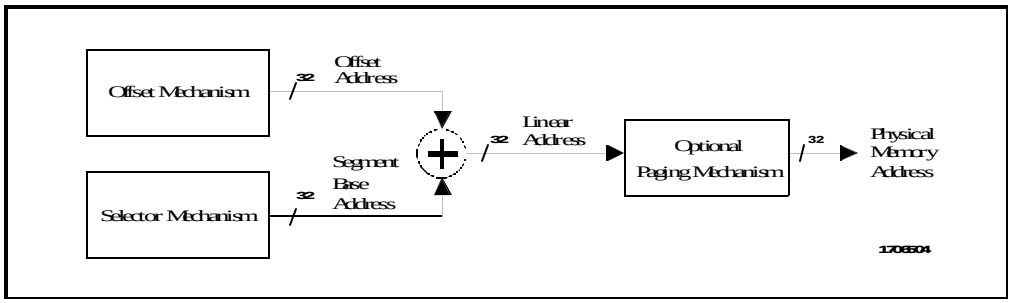


Figure 2-27. Protected Mode Address Calculation

2.8.3 Selector Mechanism

Using segmentation, memory is divided into an arbitrary number of segments, each containing usually much less than the 2^{32} byte (4 GB) maximum.

The six segment registers (CS, DS, SS, ES, FS and GS) each contain a 16-bit selector that is used when the register is loaded to locate a segment descriptor in either the global descriptor table (GDT) or the local descriptor table (LDT). The segment descriptor defines the base address, limit, and attributes of the selected segment and is cached on the Cyrix III CPU as a result of loading the selector. The cached descriptor contents are not visible to the programmer. When a memory reference occurs in protected mode, the linear address is generated by adding the segment base address in the hidden portion of the segment register to the offset address. If paging is not enabled, this linear address is used as the physical memory address. Figure 2-28 illustrates the operation of the selector mechanism.

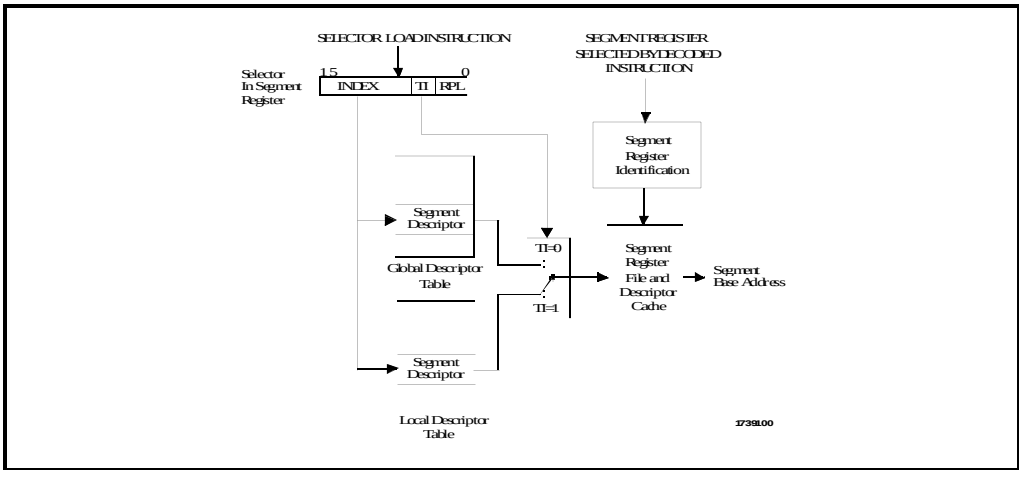


Figure 2-28. Selector Mechanism

2.8.4 Paging Mechanism

The paging mechanism translates linear addresses to their corresponding physical addresses. The page size is always 4KB. Paging is activated when the PG and the PE bits within the CR0 register are set.

The paging mechanism translates the 20 most significant bits of a linear address to a physical address. The linear address is divided into three fields DTI, PTI, PFO (, Page 2-81). These fields respectively select:

- an entry in the directory table,
- an entry in the page table selected by the directory table
- the offset in the physical page selected by the page table

The directory table and all the page tables can be considered as pages as they are 4 KB in size and are aligned on 4 KB boundaries. Each entry in these tables is 32 bits in length. The fields within the entries are detailed in Figure 2-30 (Page 2-81) and Table 2-27 (Page 2-82).

A single page directory table can address up to 4 GB of virtual memory (1,024 page tables—each table can select 1,024 pages and each page contains 4KB).

Translation Lookaside Buffer (TLB) is made up of two caches (Page 2-81).

- the L1 TLB caches page tables entries
- the L2 TLB stores PTEs that have been evicted from the L1 TLB

The L1 TLB is a 16-entry direct-mapped dual ported cache. The L2 TLB is a 384 entry, 6-way, dual ported cache.

.

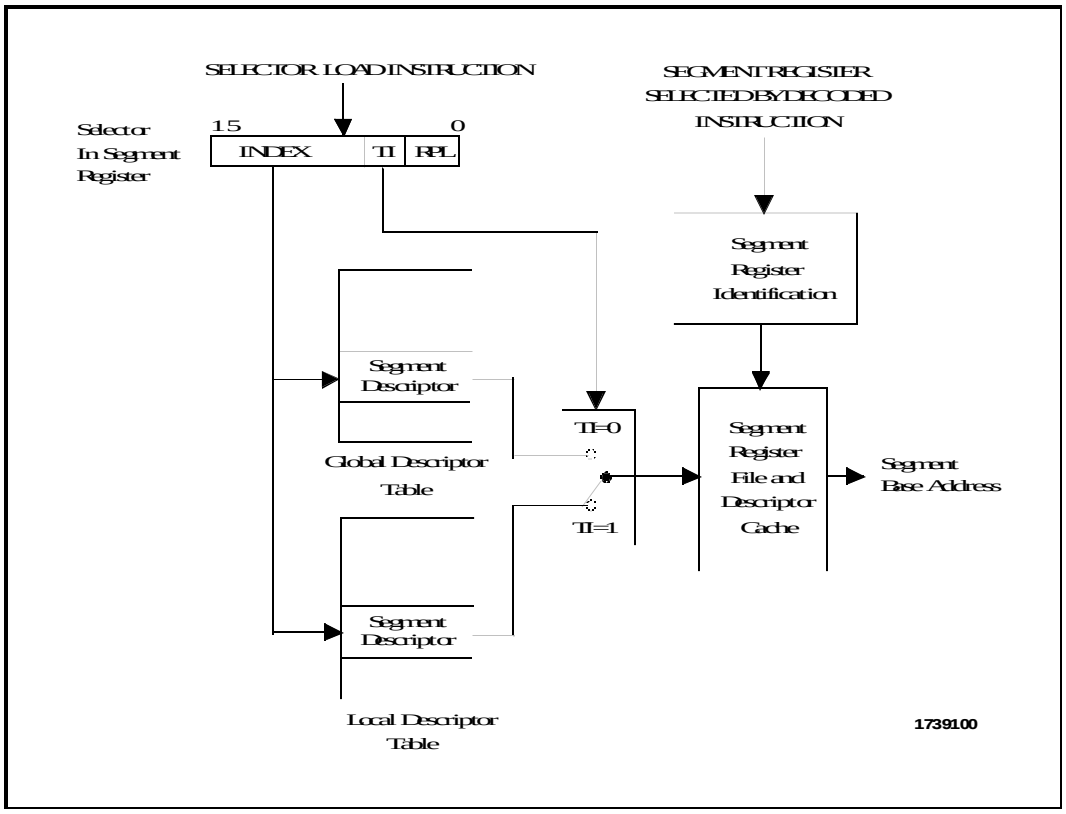


Figure 2-29. Paging Mechanism

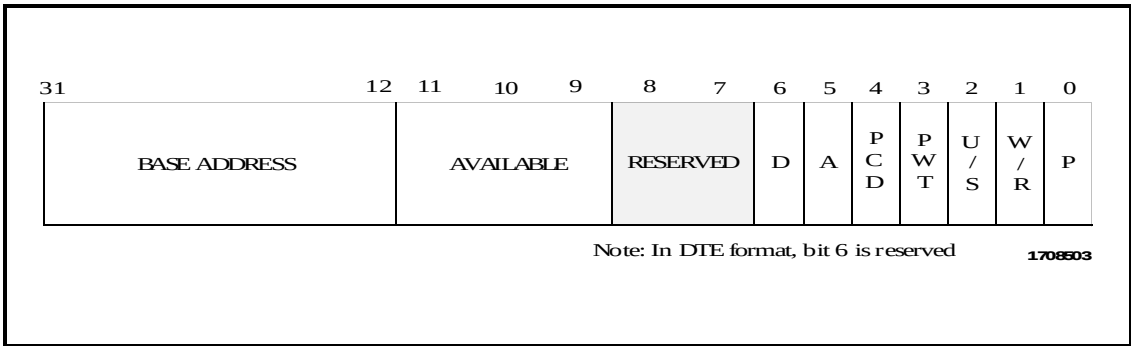


Figure 2-30. Directory and Page Table Entry (DTE and PTE) Format

Table 2-27. Directory and Page Table Entry (DTE and PTE) Bit Definitions

BIT POSITION	FIELD NAME	DESCRIPTION
31-12	BASE ADDRESS	Specifies the base address of the page or page table.
11-9	--	Undefined and available to the programmer.
8-7	--	Reserved and not available to the programmer.
6	D	Dirty Bit. If set, indicates that a write access has occurred to the page (PTE only, undefined in DTE).
5	A	Accessed Flag. If set, indicates that a read access or write access has occurred to the page.
4	PCD	Page Caching Disable Flag. If set, indicates that the page is not cacheable in the on-chip cache.
3	PWT	Page Write-Through Flag. If set, indicates that writes to the page or page tables that hit in the on-chip cache must update both the cache and external memory.
2	U/S	User/Supervisor Attribute. If set (user), page is accessible at privilege level 3. If clear (supervisor), page is accessible only when CPL ≤ 2.
1	W/R	Write/Read Attribute. If set (write), page is writable. If clear (read), page is read only.
0	P	Present Flag. If set, indicates that the page is present in RAM memory, and validates the remaining DTE/PTE bits. If clear, indicates that the page is not present in memory and the remaining DTE/PTE bits can be used by the programmer.

For a TLB hit, the TLB eliminates accesses to external directory and page tables.

The L1 TLB is a small cache optimized for speed whereas the L2 TLB is a much larger cache optimized for capacity. The L2 TLB is a proper superset of the L1 TLB.

The TLB must be flushed by the software when entries in the page tables are changed. Both the L1 and L2 TLBs are flushed whenever the CR3 register is loaded. A particular page can be flushed from the TLBs by using the INVLPG instruction.

2.8.4.1 Translation Lookaside Buffer Testing

The L1 and L2 Translation Lookaside Buffers (TLBs) can be tested by writing, then reading from the same TLB location. The operation to be performed is determined by the command (CMD) field Table 2-28 (Page 2-82) in the TR6 register.

Table 2-28. CMD Field

CMD	OPERATION	LINEAR ADDRESS BITS
x00	Write to L1	15 - 12
x01	Write to L2	17 - 12
010	Read from L1 X port	15 -12
011	Read from L2 X port	17 -12
110	Read from L1 Y port	15 - 12

Table 2-28. CMD Field

110	Read from L2 Y port	17 - 12

TLB Write

To perform a write to the Cyrix III TLBs, the TR7 register (Figure 2-31) is loaded with the desired physical address as well as the PCD and PWT bits. For a write to the L2 TLB, the SET field of TR7 must be also specified. The H1, H2, and HSET fields of TR7 are not used. The TR6 register is then loaded with the linear address, V, D, U, W and A fields and the appropriate CMD. For a L1 TLB write, the TLB entry is selected by bits 15-12 of the linear address. For a L2 TLB write, the TLB entry is selected by bits 17-12 of the linear address and the SET field of TR7.

TLB Read

For a L1 LTB read, the TR6 register is loaded with the linear address and the appropriate CMD. The L1 TLB entry selected by bits 15-12 of the linear address will then be accessed. The linear address, V, D, PG, U, W and A fields of TR6 and the physical address, PCD and PWT fields of TR7 are loaded from the specified L1 entry. The H1 bit of TR7 will indicate if the specified linear address hit in the L1 TLB.

For a L2 TLB read, the TR7 register is loaded with the desired SET. The TR6 register is then loaded with the linear address and the appropriate CMD. The L2 TLB entry selected by bits 17-12 of the linear address and the SET field in TR7 will then be accessed. The linear address, V,D, PG, V, W, and A fields of TR6 and the physical address, PCD and PWT fields of TR7 are loaded from the specified L2 entry. The H2 bit of TR7 will indicate if the specified linear address hit in the L2 TLB. If there was an L2 hit, the HSET field of TR7 will indicate which SET hit.

The TLB test register fields are defined in Table 2-29. (Page 2-84).

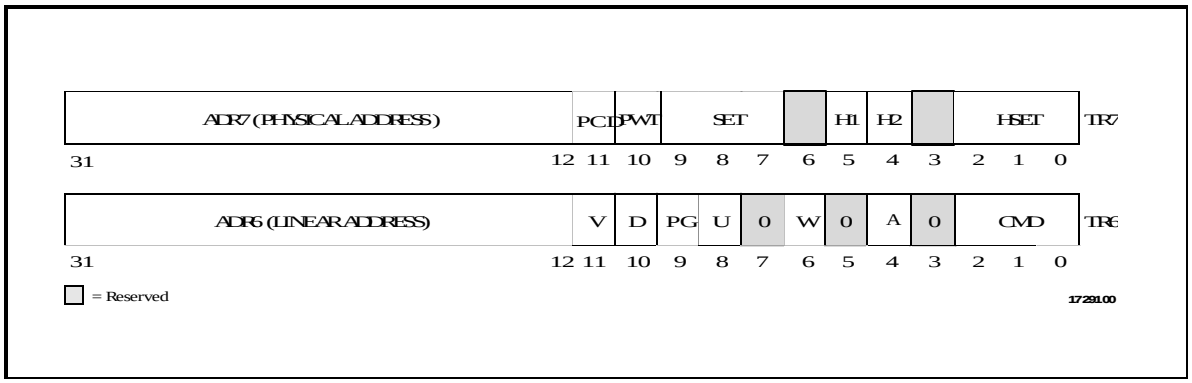


Figure 2-31. TLB Test Registers

Table 2-29. TLB Test Register Bit Definitions

REGISTER NAME	NAME	RANGE	DESCRIPTION
TR7	ADR7	31-12	Physical address or variable page size mechanism mask. TLB lookup: data field from the TLB. TLB write: data field written into the TLB.
	PCD	11	Page-level cache disable bit (PCD). Corresponds to the PCD bit of a page table entry.
	PWT	10	Page-level cache write-through bit (PWT). Corresponds to the PWT bit of a page table entry.
	SET	9-7	L2 TLB Set Selection (0h - 5h)
	H1	5	Hit in L1 TLB
	H2	4	Hit in L2 TLB
	HSET	2-0	L2 Set Selection when L2 TLB hit occurred (0h - 5h)
TR6	ADR6	31-12	Linear Address. TLB lookup: The TLB is interrogated per this address. If one and only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated per the matching TLB entry. TLB write: A TLB entry is allocated to this linear address.
	V	11	PTE Valid. TLB write: If set, indicates that the TLB entry contains valid data. If clear, target entry is invalidated.
	D	10	Dirty Attribute Bit
	PG	9	Page Global
	U	8	User/Supervisor Attribute Bit
	W	6	Write Protect bit.
	CMD	2-0	Array Command Select. Determines TLB array command. Refer to Table 2-28 (Page 2-82).

2.9 Memory Caches

The Cyrix III CPU contains two memory caches as described in Chapter 1. The Unified Cache acts as the primary data cache, and secondary instruction cache. The Instruction Line Cache is the primary instruction cache and provides a high speed instruction stream for the Integer Unit.

The unified cache is dual-ported allowing simultaneous access to any two unique banks. Two different banks may be accessed at the same time permitting any two of the following operations to occur in parallel:

- Code fetch
- Data read (X pipe, Y pipe or FPU)
- Data write (X pipe, Y pipe or FPU).

2.9.1 Unified Cache MESI States

The unified cache lines are assigned one of four MESI states as determined by MESI bits stored in tag memory. Each 32-byte cache line is divided into two 16-byte sectors. Each sector contains its own MESI bits. The four MESI states are described below:

Modified MESI cache lines are those that have been updated by the CPU, but the corresponding main memory location has not yet been updated by an external write cycle. Modified cache lines are referred to as dirty cache lines.

Exclusive MESI lines are lines that are exclusive to the Cyrix III CPU and are not duplicated within another caching agent's cache within the same system. A write to this cache line may be performed without issuing an external write cycle.

Shared MESI lines may be present in another caching agent's cache within the same system. A write to this cache line forces a corresponding external write cycle.

Invalid MESI lines are cache lines that do not contain any valid data.

2.9.1.1 Unified Cache Testing

The TR3, TR4, and TR5 on-chip test registers provide information so the unified cache can be tested. This information determines what particular area will be tested. Fields within these test registers identify which area of the cache will be selected for testing.

Cache Organization. The unified cache (Figure 2-32) is divided into 32-bytes lines. This cache is divided into four sets. Since a set (as well as the cache) is smaller than main memory, each line in the set corresponds to more than one line in main memory. When a cache line is allocated, bits A31-A14 of the main memory address are stored in the cache line tag. The remaining address bits are used to identify the specific 32-byte cache line (A13-A5), and the specific 4-byte entry within the cache line (A4-A2).

Test Initiation. A test register operation is initiated by writing to the TR5 register shown in Figure 2-33 (Page 2-87) using a special MOV instruction. The TR5 CTL field, detailed in Table 2-30 (Page 2-87), determines the function to be performed. For cache writes, the registers TR4 and TR3 must be initialized before a write is made to TR5. Eight 4-byte accesses are required to access a complete cache line.

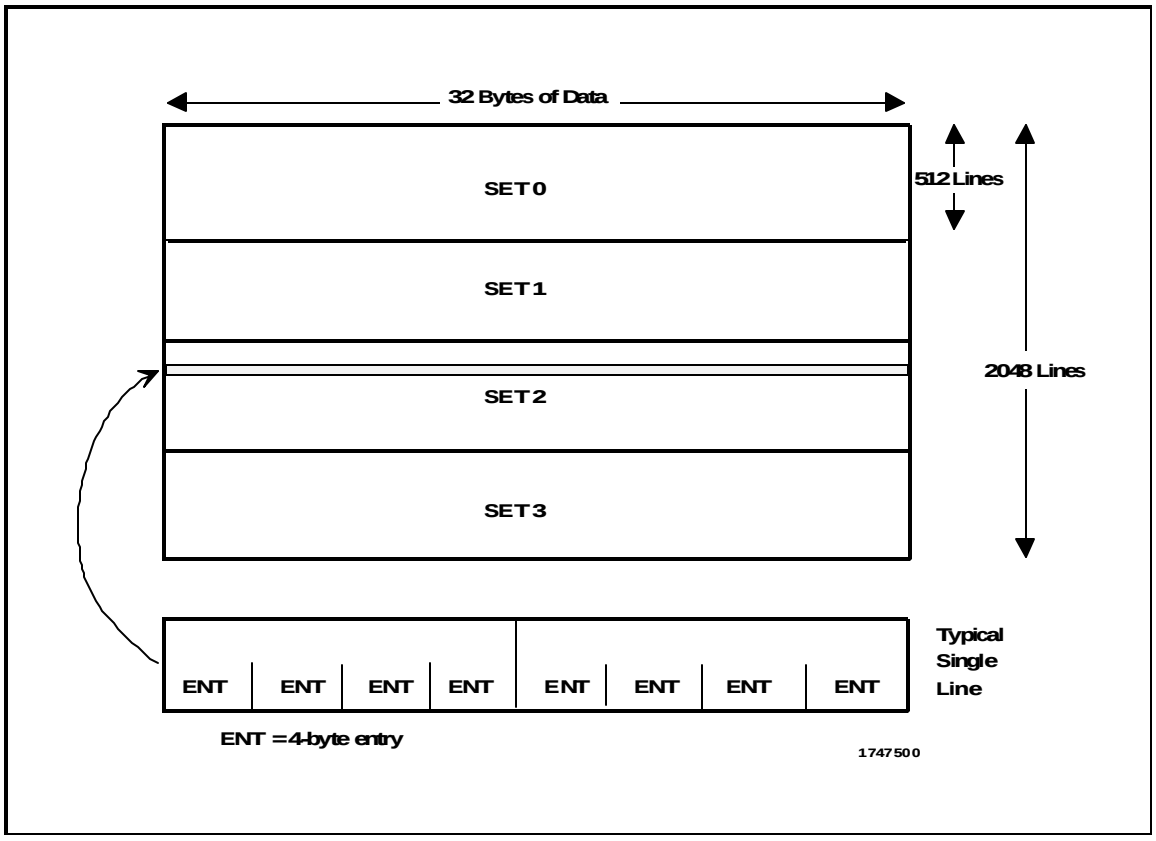


Figure 2-32. Unified Cache

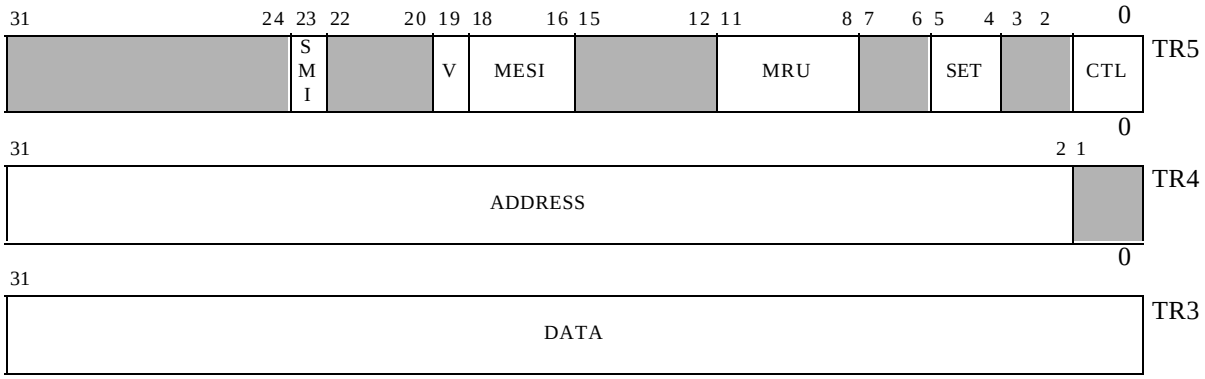


Figure 2-33. Cache Test Registers

Table 2-30. Cache Test Register Bit Definitions

REGISTER NAME	FIELD NAME	RANGE	DESCRIPTION
TR5	SMI	23	SMI Address Bit. Selects separate/cacheable SMI code/data space
	V, MESI	19 - 16	Valid, MESI Bits* If = 1000, Modified If = 1001, Shared If = 1010, Exclusive If = 0011, Invalid If = 1100, Locked Valid If = 0111, Locked Invalid Else = Undefined
	MRU	11 - 8	Used to determine the Least Recently Used (LRU) line.
	SET	5 - 4	Cache Set. Selects one of four cache sets to perform operation on.
	CTL	1 - 0	Control field If = 00: flush cache without invalidate If = 01: write cache If = 10: read cache If = 11: no cache or test register modification
TR4	ADDRESS	31 - 2	Physical Address
TR3	DATA	31 - 0	Data written or read during a cache test.

*Note: All 32 bytes should contain valid data before a line is marked as valid.

Write Operations. During a write, the TR3 DATA (32-bits) and TAG field information is written to the address selected by the ADDRESS field in TR4 and the SET field in TR5.

Read Operations. During a read, the cache address selected by the ADDRESS field in TR4 and the SET field in TR5. The TVB, MESI and MRU fields in TR5 are updated with the information from the selected line. TR3 holds the selected read data.

Cache Flushing. A cache flush occurs during a TR5 write if the CTL field is set to zero. During flushing, the CPU's cache controller reads through all the lines in the cache. "Modified" lines are redefined as "shared" by setting the shared MESI bit. Clean lines are left in their original state.

2.9.2 RAM Cache Locking

RAM cache locking (was called Scratch Pad Memory) sets up a private area of memory that can be assigned within the Cyrix III unified cache. Cached locked RAM is read/writable and is NOT kept coherent with the rest of the system. Scratch Pad Memory is a separate memory on certain Cyrix CPUs.

Cache locking may be implemented differently on different processors. On the Cyrix III CPU, the cache locking RAM may be assigned on a cache line granularity.

RDMSR and WRMSR instructions (Page 2-39) with indices 03h to 05h are used to assign scratch pad memory. These instructions access the cache test registers. See section 2.9.1.1 (Page 2-86) for detailed description of cache test register operation. The cache line is assigned into Scratch Pad RAM by setting its MESI state to “locked valid.”

When locking physical addresses into the cache (Table 2-31), the programmer should be aware of several issues:

1) Locking all sets of the cache should not be done. It is required that one set always be available for general purpose caching. 2) Care must be taken by the programmer not to create synonyms. This is done by first checking to see if a particular address is locked before attempting to lock the address. If synonyms are created, Cyrix III CPU operation will be undefined.

When ever possible, it is recommended to flush the cache before assigning locked memory areas. Locked areas of the cache are cleared on reset, and are unaffected by warm reset and FLUSH#, or the INVD and WBINVD instructions.

Table 2-31. RAM Cache Locking Operations

Read/Write	ECX	EDX	EAX	Operation
Read/Write	03h	----	Data to be read or written from/to the cache.	Loads or stores data to/from TR3.
Write	04h	----	32 bits of address	Address in EAX is loaded into TR4. This address is the cache line address that will be locked.
Read	04h	----	32 bits of address	Stores the contents of TR4 in EAX
Write	05h	----	Data to be written into TR5	Performs operation specified in CTL field of TR5.
Read	05h	----	Data in TR5 register	Reads data in TR5 and stores in EAX.

2.10 Interrupts and Exceptions

The processing of an interrupt or an exception changes the normal sequential flow of a program by transferring program control to a selected service routine. Except for SMM interrupts, the location of the selected service routine is determined by one of the interrupt vectors stored in the interrupt descriptor table.

Hardware interrupts are generated by signal sources external to the CPU. All exceptions (including so-called software interrupts) are produced internally by the CPU.

2.10.1 Interrupts

External events can interrupt normal program execution by using one of the three interrupt pins on the Cyrix III CPU.

- Non-maskable Interrupt (NMI pin)
- Maskable Interrupt (INTR pin)
- SMM Interrupt (SMI# pin).

For most interrupts, program transfer to the interrupt routine occurs after the current instruction has been completed. When the execution returns to the original program, it begins immediately following the last completed instruction.

With the exception of string operations, interrupts are acknowledged between instructions. Long string operations have interrupt windows between memory moves that allow interrupts to be acknowledged.

The NMI interrupt cannot be masked by software and always uses interrupt vector 2 to locate its service routine. Since the interrupt vector is fixed and is supplied internally, no interrupt acknowledge bus cycles are performed. This interrupt is normally reserved for unusual situations such as parity errors and has priority over INTR interrupts.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If NMI is re-asserted prior to execution of the IRET instruction, one and only one NMI rising edge is stored and processed after execution of the next IRET. During the NMI service routine, maskable interrupts may be enabled (unmasked). If an unmasked INTR occurs during the NMI service routine, the INTR is serviced and execution returns to the NMI service routine following the next IRET. If a HALT instruction is executed within the NMI service routine, the Cyrix III CPU restarts execution only in response to RESET#, an unmasked INTR or an SMM interrupt. NMI does not restart CPU execution under this condition.

The INTR interrupt is unmasked when the Interrupt Enable Flag (IF) in the EFLAGS register is set to 1. When an INTR interrupt occurs, the CPU performs two locked interrupt acknowledge bus cycles. During the second cycle, the CPU reads an 8-bit vector that is supplied by an external interrupt controller. This vector selects one of the 256 possible interrupt handlers which will be executed in response to the interrupt.

The SMM interrupt has higher priority than either INTR or NMI. After SMI# is asserted, program execution is passed to an SMI service routine that runs in SMM address space reserved for this purpose. The remainder of this section does not apply to the SMM interrupts. SMM interrupts are described in greater detail later in this chapter.

2.10.2 Exceptions

Exceptions are generated by an interrupt instruction or a program error. Exceptions are classified as traps, faults or aborts depending on the mechanism used to report them and the restart ability of the instruction that first caused the exception.

A Trap Exception is reported immediately following the instruction that generated the trap exception. Trap exceptions are generated by execution of a software interrupt instruction (INTO, INT 3, INT n, BOUND), by a single-step operation or by a data breakpoint.

Software interrupts can be used to simulate hardware interrupts. For example, an INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table. Execution of the interrupt service routine occurs regardless of the state of the IF flag in the EFLAGS register.

The one byte INT 3, or breakpoint interrupt (vector 3), is a particular case of the INT n instruction. By inserting this one byte instruction in a program, the user can set breakpoints in the code that can be used during debug.

Single-step operation is enabled by setting the TF bit in the EFLAGS register. When TF is set, the CPU generates a debug exception (vector 1) after the execution of every instruction. Data breakpoints also generate a debug exception and are specified by loading the debug registers (DR0-DR7) with the appropriate values.

A Fault Exception is reported prior to completion of the instruction that generated the exception. By reporting the fault prior to instruction completion, the CPU is left in a state that allows the instruction to be restarted and the effects of the faulting instruction to be nullified. Fault exceptions include divide-by-zero errors, invalid opcodes, page faults and coprocessor errors. Instruction breakpoints (vector 1) are also handled as faults. After execution of the fault service routine, the instruction pointer points to the instruction that caused the fault.

An Abort Exception is a type of fault exception that is severe enough that the CPU cannot restart the program at the faulting instruction. The double fault (vector 8) is the only abort exception that occurs on the Cyrix III CPU.

2.10.3 Interrupt Vectors

When the CPU services an interrupt or exception, the current program's FLAGS, code segment and instruction pointer are pushed onto the stack to allow resumption of execution of the interrupted program. In protected mode, the processor also saves an error code for some exceptions. Program control is then transferred to the interrupt handler (also called the interrupt service routine). Upon execution of an IRET at the end of the service routine, program execution resumes by popping from the stack, the instruction pointer, code segment, and FLAGS.

Interrupt Vector Assignments

Each interrupt (except SMI#) and exception is assigned one of 256 interrupt vector numbers Table 2-32, (Page 2-93). The first 32 interrupt vector assignments are defined or reserved. INT instructions acting as software interrupts may use any of the interrupt vectors, 0 through 255.

Table 2-32. Interrupt Vector Assignments

INTERRUPT VECTOR	FUNCTION	EXCEPTION TYPE
0	Divide error	FAULT
1	Debug exception	TRAP/FAULT*
2	NMI interrupt	
3	Breakpoint	TRAP
4	Interrupt on overflow	TRAP
5	BOUND range exceeded	FAULT
6	Invalid opcode	FAULT
7	Device not available	FAULT
8	Double fault	ABORT
9	Reserved	
10	Invalid TSS	FAULT
11	Segment not present	FAULT
12	Stack fault	FAULT
13	General protection fault	TRAP/FAULT
14	Page fault	FAULT
15	Reserved	
16	FPU error	FAULT
17	Alignment check exception	FAULT
18-31	Reserved	
32-255	Maskable hardware interrupts	TRAP
0-255	Programmed interrupt	TRAP

*Note: Data breakpoints and single-steps are traps. All other debug exceptions are faults.

In response to a maskable hardware interrupt (INTR), the Cyrix III CPU issues an interrupt acknowledge bus cycle to read the vector number from external hardware. These vectors should be in the range 32 - 255 as vectors 0 - 31 are reserved.

Interrupt Descriptor Table

The interrupt vector number is used by the Cyrix III CPU to locate an entry in the interrupt descriptor table (IDT). In real mode, each IDT entry consists of a four-byte far pointer to the beginning of the corresponding interrupt service routine. In protected mode, each IDT entry is an eight-byte descriptor. The Interrupt Descriptor Table Register (IDTR) specifies the beginning address and limit of the IDT. Following reset, the IDTR contains a base address of 0h with a limit of 3FFh.

The IDT can be located anywhere in physical memory as determined by the IDTR register. The IDT may contain different types of descriptors: interrupt gates, trap gates and task gates. Interrupt gates are used primarily to enter a hardware interrupt handler. Trap gates are generally used to enter an exception handler or software interrupt handler. If an interrupt gate is used, the Interrupt Enable Flag (IF) in the EFLAGS register is cleared before the interrupt handler is entered. Task gates are used to make the transition to a new task.

2.10.4 Interrupt and Exception Priorities

As the Cyrix III CPU executes instructions, it follows a consistent policy for prioritizing exceptions and hardware interrupts. The priorities for competing interrupts and exceptions are listed in Table 2-33 (Page 2-95). Debug traps for the previous instruction and the next instructions always take precedence. SMM interrupts are the next priority. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the Cyrix III processor services the NMI interrupt first.

The Cyrix III CPU checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result from a single instruction. However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should make the appropriate corrections to the instruction and then restart the instruction. In this way, exceptions can be serviced until the instruction executes properly.

The Cyrix III CPU supports instruction restart after all faults, except when an instruction causes a task switch to a task whose task state segment (TSS) is partially not present. A TSS can be partially not present if the TSS is not page aligned and one of the pages where the TSS resides is not currently in memory

.

Table 2-33. Interrupt and Exception Priorities

PRIORITY	DESCRIPTION	NOTES
0	Warm Reset	Caused by the assertion of INIT#.
1	Debug traps and faults from previous instruction.	Includes single-step trap and data breakpoints specified in the debug registers.
2	Debug traps for next instruction.	Includes instruction execution breakpoints specified in the debug registers.
3	Hardware Cache Flush	Caused by the assertion of FLUSH#.
4	SMM hardware interrupt.	SMM interrupts are caused by SMI# asserted and always have highest priority.
5	Non-maskable hardware interrupt.	Caused by NMI asserted.
6	Maskable hardware interrupt.	Caused by INTR asserted and IF = 1.
7	Faults resulting from fetching the next instruction.	Includes segment not present, general protection fault and page fault.
8	Faults resulting from instruction decoding.	Includes illegal opcode, instruction too long, or privilege violation.
9	WAIT instruction and TS = 1 and MP = 1.	Device not available exception generated.
10	ESC instruction and EM = 1 or TS = 1.	Device not available exception generated.
11	Floating point error exception.	Caused by unmasked floating point exception with NE = 1.
12	Segmentation faults (for each memory reference required by the instruction) that prevent transferring the entire memory operand.	Includes segment not present, stack fault, and general protection fault.
13	Page Faults that prevent transferring the entire memory operand.	
14	Alignment check fault.	

2.10.5 Exceptions in Real Mode

Many of the exceptions described in Table 2-33 (Page 2-95) are not applicable in real mode. Exceptions 10, 11, and 14 do not occur in real mode. Other exceptions have slightly different meanings in real mode as listed in Table 2-34.

Table 2-34. Exception Changes in Real Mode

VECTOR NUMBER	PROTECTED MODE FUNCTION	REAL MODE FUNCTION
8	Double fault.	Interrupt table limit overrun.
10	Invalid TSS.	x
11	Segment not present.	x
12	Stack fault.	SS segment limit overrun.
13	General protection fault.	CS, DS, ES, FS, GS segment limit overrun.
14	Page fault.	x

Note: x = does not occur

2.10.6 Error Codes

When operating in protected mode, the following exceptions generate a 16-bit error code:

Double Fault	Invalid TSS
Alignment Check	Segment Not Present
Page Fault	Stack Fault
	General Protection Fault

The error code is pushed onto the stack prior to entering the exception handler. The error code format is shown in Figure 2-34 and the error code bit definitions are listed in Table 2-35. Bits 15-3 (selector index) are not meaningful if the error code was generated as the result of a page fault. The error code is always zero for double faults and alignment check exceptions.

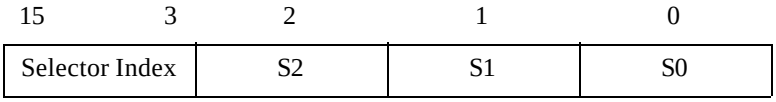


Figure 2-34. Error Code Format

Table 2-35. Error Code Bit Definitions

FAULT TYPE	SELECTOR INDEX (BITS 15-3)	S2 (BIT 2)	S1 (BIT 1)	S0 (BIT 0)
Double Fault or Alignment Check	0	0	0	0
Page Fault	Reserved.	Fault caused by: 0 = not present page 1 = page-level protection violation.	Fault occurred during: 0 = read access 1 = write access.	Fault occurred during: 0 = supervisor access. 1 = user access.
IDT Fault	Index of faulty IDT selector.	Reserved.	1	If = 1, exception occurred while trying to invoke exception or hardware interrupt handler.
Segment Fault	Index of faulty selector.	TI bit of faulty selector.	0	If =1, exception occurred while trying to invoke exception or hardware interrupt handler.

2.11 System Management Mode

System Management Mode (SMM) is a distinct CPU mode that differs from normal CPU x86 operating modes (real mode, V86 mode, and protected mode) and is most often used to perform power management.

The Cyrix III CPU is backward compatible with the SL-compatible SMM found on previous Cyrix microprocessors. On the Cyrix III SMM has been enhanced to optimized software emulation of multimedia and I/O peripherals.

The Cyrix Enhanced SMM provides new features:

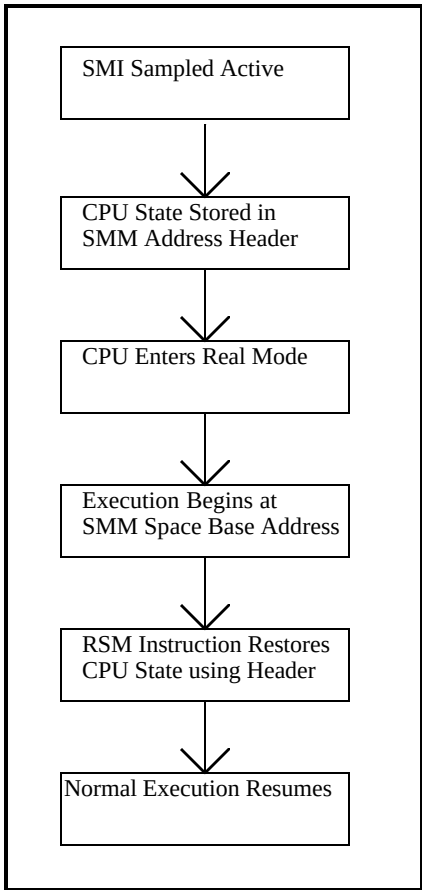
- Cacheability of SMM memory
- Improved SMM entry and exit time.

Overall Operation

The overall operation of a SMM operation is shown in (Figure 2-35). SMM is entered using the System Management Interrupt (SMI) pin. SMI interrupts have higher priority than any other interrupt, including NMI interrupts.

Upon entering SMM mode, portions of the CPU state are automatically saved in the SMM address memory space header. The CPU enters real mode and begins executing the SMI service routine in SMM address space.

Execution of a SMM routine starts at the base address in SMM memory address space. Since the SMM routines reside in SMM memory space, SMM routines can be made totally transparent to all software, including protected-mode operating systems.



SMI Execution Flow Diagram

Figure 2-35. SMI Execution Flow Diagram

2.11.1 SMM Memory Space

SMM memory must reside within the bounds of physical memory and not overlap with system memory. SMM memory space (Figure 2-36) is defined by setting the SM3 bit in CCR1 and specifying the base address and size of the SMM memory space in the ARR3 register.

The base address must be a multiple of the SMM memory space size. For example, a 32 KB SMM memory space must be located on a 32KB address boundary. The memory space size can range from 4 KB to 4 GB SMM accesses ignore the state of the A20M# input pin and drive the A20 address bit to the unmasked value.

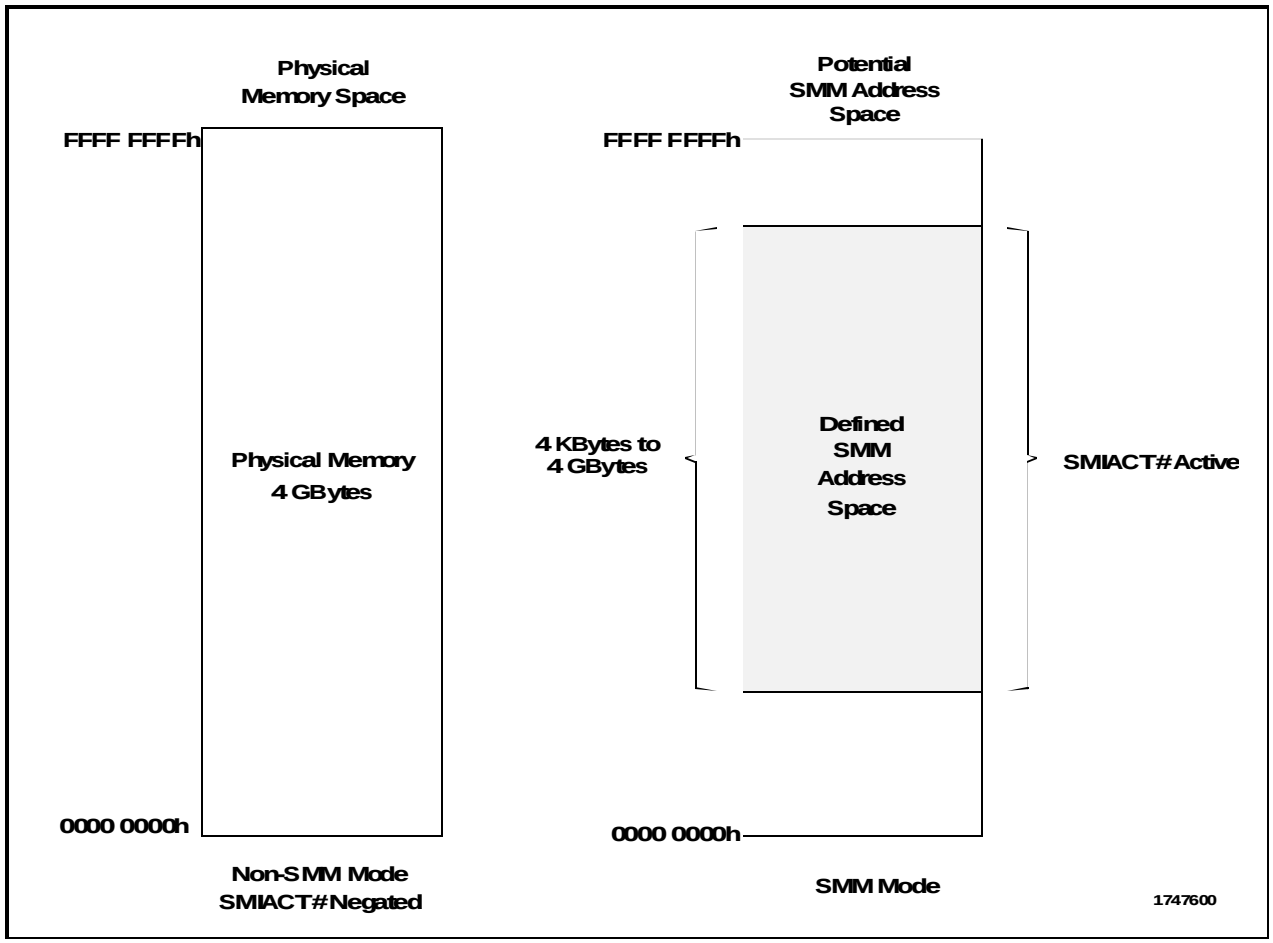


Figure 2-36. System Management Memory Space

2.11.2 SMM Memory SpaceHeader

The SMM Memory Space Header (Figure 2-37) is used to store the CPU state prior to starting an SMM routine. The fields in this header are described in Table 2-36 (Page 2-101). After the SMM routine has completed, the header information is used to restore the original CPU state. The location of the SMM header is determined by the SMM Header Address Register (SMHR).

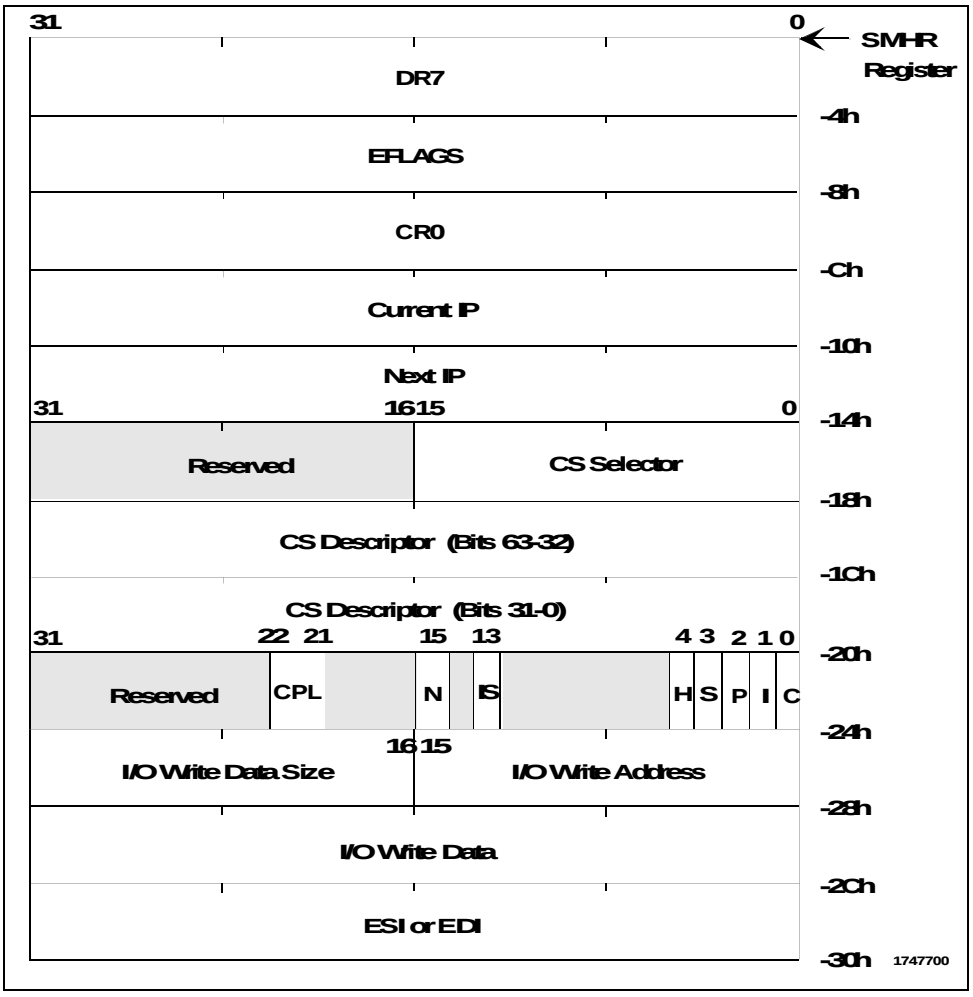


Figure 2-37. SMM Memory Space Header

Table 2-36. SMM Memory Space Header

NAME	DESCRIPTION	SIZE
DR7	The contents of Debug Register 7.	4 Bytes
EFLAGS	The contents of Extended Flags Register.	4 Bytes
CR0	The contents of Control Register 0.	4 Bytes
Current IP	The address of the instruction executed prior to servicing SMI interrupt.	4 Bytes
Next IP	The address of the next instruction that will be executed after exiting SMM mode.	4 Bytes
CS Selector	Code segment register selector for the current code segment.	2 Bytes
CS Descriptor	Code segment register descriptor for the current code segment.	8 Bytes
CPL	Current privilege level for current code segment.	2 Bits
IS	Internal SMI Indicator If IS =1: current SMM is the result of an internal SMI event. If IS =0: current SMM is the result of an external SMI event.	1 Bit
H	SMI during CPU HALT state indicator If H = 1: the processor was in a halt or shutdown prior to servicing the SMM interrupt.	1 Bit
S	Software SMM Entry Indicator. If S = 1: current SMM is the result of an SMINT instruction. If S = 0: current SMM is not the result of an SMINT instruction.	1 Bit
P	REP INSx/OUTSx Indicator If P = 1: current instruction has a REP prefix. If P = 0: current instruction does not have a REP prefix.	1 Bit
I	IN, INSx, OUT, or OUTSx Indicator If I = 1: if current instruction performed is an I/O WRITE. If I = 0: if current instruction performed is an I/O READ.	1 Bit
C	Code Segment writable Indicator If C = 1: the current code segment is writable. If C = 0: the current code segment is not writable.	1 Bit
I/O	Indicates size of data for the trapped I/O write: 01h = byte 03h = word 0Fh = dword	2 Bytes
I/O Write Address	I/O Write Address Processor port used for the trapped I/O write.	2 Bytes
I/O Write Data	I/O Write Data Data associated with the trapped I/O write.	4 Bytes
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap.	4 Bytes

Note: INSx = INS, INSB, INSW or INSD instruction.
Note: OUTSx = OUTS, OUTSB, OUTSW and OUTSD instruction.

Current and Next IP Pointers

Included in the header information are the Current and Next IP pointers. The Current IP points to the instruction executing when the SMI was detected and the Next IP points to the instruction that will be executed after exiting SMM.

Normally after an SMM routine is completed, the instruction flow begins at the Next IP address. However, if an I/O trap has occurred, instruction flow should return to the Current IP to complete the I/O instruction.

If SMM has been entered due to an I/O trap for a REP INSx or REP OUTSx instruction, the Current IP and Next IP fields contain the same address.

If an entry into SMM mode was caused by an I/O trap, the port address, data size and data value associated with that I/O operation are stored in the SMM header. Note that these values are only valid for I/O operations. The I/O data is not restored within the CPU when executing a RSM instruction.

Under these circumstances the I and P bits, as well as ESI/EDI field, contain valid information.

Also saved are the contents of debug register 7 (DR7), the extended flags register (EFLAGS), and control register 0 (CR0).

SMM Header Address Pointer

The SMM Header Address Pointer Register (SMHR) (Figure 2-38) contains the 32-bit SMM Header pointer. The SMHR address is dword aligned, so the two least significant bits are ignored.

The SMHR valid bit (bit 0) is cleared with every write to ARR3 and during a hardware RESET#. Upon entry to SMM, the SMHR valid bit is examined before the CPU state is saved into the SMM memory space header. When the valid bit is reset, the SMM header pointer will be calculated (ARR3 base field + ARR3 size field) and loaded into the SMHR and the valid bit will be set.

If the desired SMM header location is different than the top of SMM memory space, as may be the case when nesting SMI's, then the SMHR register must be loaded with a new value and valid bit from within the SMI routine before nesting is enabled.

The SMM memory space header can be relocated using the new RDSHR and WRSHR instructions.

Figure 2-38. SMHR Register

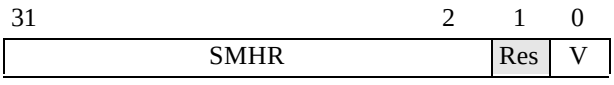


Table 2-37. SMHR Register Bits

BIT POSITION	DESCRPTION
31 - 2	SMHR header pointer address.
1	Reserved
0	Valid Bit

2.11.3 SMM Instructions

After entering the SMI service routine, the MOV, SVDC, SVLDT and SVTS instructions (Table 2-38) can be used to save the complete CPU state information. If the SMI service routine modifies more than what is automatically saved or forces the CPU to power down, the complete CPU state information must be saved. Since the CPU is a static device, its internal state is retained when the input clock is stopped. Therefore, an entire CPU state save is not necessary prior to stopping the input clock.

Table 2-38. SMM Instruction Set

INSTRUCTION	OPCODE	FORMAT	DESCRIPTION
SVDC	0F 78 [mod sreg3 r/m]	SVDC mem80, sreg3	*Save Segment Register and Descriptor* Saves reg (DS, ES, FS, GS, or SS) to mem80.
RSDC	0F 79 [mod sreg3 r/m]	RSDC sreg3, mem80	*Restore Segment Register and Descriptor* Restores reg (DS, ES, FS, GS, or SS) from mem80. Use RSM to restore CS. Note: Processing "RSDC CS, Mem80" will produce an exception.
SVLDT	0F 7A [mod 000 r/m]	SVLDT mem80	*Save LDTR and Descriptor* Saves Local Descriptor Table (LDTR) to mem80.
RSLDT	0F 7B [mod 000 r/m]	RSLDT mem80	*Restore LDTR and Descriptor* Restores Local Descriptor Table (LDTR) from mem80.
SVTS	0F 7C [mod 000 r/m]	SVTS mem80	*Save TSR and Descriptor* Saves Task State Register (TSR) to mem80.
RSTS	0F 7D [mod 000 r/m]	RSTS mem80	*Restore TSR and Descriptor* Restores Task State Register (TSR) from mem80.
SMINT	0F 38	SMINT	*Software SMM Entry* CPU enters SMM mode. CPU state information is saved in SMM memory space header and execution begins at SMM base address.
RSM	0F AA	RSM	*Resume Normal Mode* Exits SMM mode. The CPU state is restored using the SMM memory space header and execution resumes at interrupted point.
RDSHR	0F 36	RDSHR ereg/mem32	*Read SMM Header Pointer Register* Saves SMM header pointer to extended register or memory.
WRSHR	0F 37	WRSHR ereg/mem32	*Write SMM Header Pointer Register* Load SMM header pointer register from extended register or memory.

Note: mem32 = 32-bit memory location
mem80 = 80-bit memory location

The SMM instructions listed in Table 2-38, (except the SMINT instruction) can be executed only if:

1) ARR3 Size > 0
2) Current Privilege Level =0
3) the CPU is executing an SMI service routine.
4) USE_SMI (CCR1- bit 1) = 1
5) SM3 (CCR1-bit 7) = 1

If the above conditions are not met and an attempt is made to execute an SVDC, RSDC, SVLDT, RSLDT, SVTS, RSTS, SMINT, RSM, RDSHR, or WDSHR instruction, an invalid opcode exception is generated. These instructions can be executed outside of defined SMM space provided the above conditions are met.

The SVDC, RSDC, SVLDT, RSLDT, SVTS and RSTS instructions save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

The WRSHR instruction loads the contents of either a 32-bit memory operand or a 32-bit register operand into the SMHR pointer register based on the value of the mod r/m instruction byte. Likewise the RDSHR instruction stores the contents of the SMHR pointer register to either a 32 bit memory operand or a 32 bit register operand based on the value of the mod r/m instruction byte.

2.11.4 SMM Operation

This section details the SMM operations.

Entering SMM

Entering SMM requires the assertion of the SMI# pin. SMI interrupts have higher priority than any interrupt including NMI interrupts.

For the SMI# to be recognized, the following configuration register bits must be set as shown in Table 2-39.

Table 2-39. Requirements for Recognizing SMI# and SMINT

REGISTER (Bit)		SMI#	SMINT
SMI	CCR1 (1)	1	1
ARR3	SIZE (3-0)	> 0	> 0
SM3	CCR1 (7)	1	1

Upon entry into SMM, after the SMM header has been saved, the CR0, EFLAGS, and DR7 registers are set to their reset values. The Code Segment (CS) register is loaded with the base, as defined by the ARR3 register, and a limit of 4 GB. The SMI service routine then begins execution at the SMM base address in real mode.

Saving the CPU State

The programmer must save the value of any registers that may be changed by the SMI service routine. For data accesses immediately after entering the SMI service routine, the programmer must use CS as a segment override. I/O port access is possible during the routine but care must be taken to save registers modified by the I/O instructions. Before using a segment register, the register and the register's descriptor cache contents should be saved using the SVDC instruction. While executing in the SMM space, execution flow can transfer to normal memory locations.

Program Execution

Hardware interrupts, (INTRs and NMIs), may be serviced during a SMI service routine. If interrupts are to be serviced while executing in the SMM memory space, the SMM memory space must be within the 0 to 1 MB address range to guarantee proper return to the SMI service routine after handling the interrupt.

INTRs are automatically disabled when entering SMM since the IF flag is set to its reset value. Once in SMM, the INTR can be enabled by setting the IF flag. NMI is also automatically disable when entering SMM. Once in SMM, NMI can be enabled by setting NMI_EN in CCR3. If NMI is not enabled, the CPU latches one NMI event and services the interrupt after NMI has been enabled or after exiting SMM through the RSM instruction.

Within the SMI service routine, protected mode may be entered and exited as required, and real or protected mode device drivers may be called.

Exiting SMM

To exit the SMI service routine, a Resume (RSM) instruction, rather than an IRET, is executed. The RSM instruction causes the Cyrix III processor to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the full CPU state was saved by the programmer, the stored values should be reloaded prior to executing the RSM instruction using the MOV, RSDC, RSLDT and RSTS instructions.

When the RSM instruction is executed at the end of the SMI handler, the EIP instruction pointer is automatically read from the NEXT IP field in the SMM header.

When restarting I/O instructions, the value of NEXTIP may need modification. Before executing the RSM instruction, use a MOV instruction to move the CURRENTIP value to the NEXT IP location as the CURRENT IP value is valid if an I/O instruction was executing when the SMI interrupt occurred. Execution is then returned to the I/O instruction, rather than to the instruction after the I/O instruction.

A set H bit in the SMM header indicates that a HLT instruction was being executed when the SMI occurred. To resume execution of the HLT instruction, the NEXTIP field in the SMM header should be decremented by one before executing RSM instruction.

2.11.5 SL and Cyrix SMM Operating Modes

There are two SMM modes, SL-compatible mode (default) and Cyrix SMM mode.

2.11.5.1 SL-Compatible SMM Mode

While in SL-compatible mode, SMM memory space accesses can only occur during an SMI service routine. This includes the time when the SMI service routine accesses memory outside the defined SMM memory space.

SMM memory caching is not supported in SL-compatible SMM mode. If a cache inquiry cycle occurs while in SMM mode, any resulting write-back cycle is issued with SMM_MEM asserted. This occurs even though the write-back cycle is intended for normal memory rather than SMM memory. To avoid this problem it is recommended that the internal caches be flushed prior to servicing an SMI event. Of course in write-back mode this could add an indeterminate delay to servicing of SMI.

An interrupt on the SMI# input pin has higher priority than the NMI input. The SMI# input pin is falling edge sensitive and is sampled on every rising edge of the processor input clock.

Asserting SMI# forces the processor to save the CPU state to memory defined by SMHR register and to begin execution of the SMI service routine at the beginning of the defined SMM memory space. After the processor internally acknowledges the SMI# interrupt, an SMM acknowledge cycle is driven onto the bus, and SMM_MEM (pin EX4) is asserted.

When the RSM instruction is executed, the CPU negates the SMM_MEM pin after the last bus cycle to SMM memory. While executing the SMM service routine, one additional SMI# can be latched for service after resuming from the first SMI.

During RESET#, the USE_SMI bit in CCR1 is cleared.

2.11.5.2 Cyrix Enhanced SMM Mode

The Cyrix SMM Mode is enabled when bit0 in the CCR6 (SMM_MODE) is set. Only in Cyrix enhanced SMM mode can SMM space be cached.

Cacheability of SMM Space

In SL-compatible SMM mode, caching is not available, but in Cyrix SMM mode, both code and data caching is supported. In order to cache SMM data and avoid coherency issues the processor assumes no overlap of main memory with SMM memory. This implies that a section of main memory must be dedicated for SMM.

The on-chip cache sets a special ID bit in the cache tag block for each line that contains SMM code data. This ID bit is then used by the bus controller to regulate assertion of the SMM_MEM pin for write-back of any SMM data.

2.11.6 Maintaining the FPU and MMX States

If power will be removed from the CPU or if the SMM routine will execute MMX or FPU instructions, then the MMX or FPU state should

be maintained for the application running before SMM was entered. If the MMX or FPU state is to be saved and restored from within SMM, there are certain guidelines that must be followed to make SMM completely transparent to the application program.

The complete state of the FPU can be saved and restored with the FNSAVE and FNRSTOR instructions. FNSAVE is used instead of the FSAVE because FSAVE will wait for the FPU to check for existing error conditions before storing the FPU state. If there is a unmasked FPU exception condition pending, the FSAVE instruction will wait until the exception condition is serviced. To maintain transparency for the application program, the SMM routine should not service this exception. If the FPU state is restored with the FNRSTOR instruction before returning to normal mode, the application program can correctly service the exception. FPU instructions can be executed within SMM once the FPU state has been saved.

The information saved with the FSAVE instruction varies depending on the operating mode of the CPU. To save and restore all FPU information, the 32-bit protected mode version of the FPU save and restore instruction should be used.

2.12 Sleep and Halt

The Halt Instruction (HLT) stops program execution and prevents the processor from using the local bus until restarted. The Cyrix III CPU then issues a special HALT bus cycle and enters a low-power suspend mode if the SUSP_HLT bit in CCR2 is set. SMI, NMI, INTR with interrupts enabled (IF bit in EFLAGS=1), INIT# or RESET# force the CPU out of the halt state. If interrupted, the saved code segment and instruction pointer specify the instruction following the HLT.

Sleep states can be entered using STPCLK# and SLP# pins to perform power management. The three low power states are Stop Grant, Sleep, and Deep Sleep.

Stop Grant state is entered by asserting STPCLK#. When STPCLK# is deasserted, the Stop Grant state is exited and processor returns to normal operation.

Sleep state is entered by asserting the SLP# pin while the processor is in Stop Grant state. Sleep state is exited when SLP# is deasserted and the processor returns to Stop Grant state.

Deep Sleep can be entered while in Sleep state by stopping the bus clock pin BCLK. When BCLK is restarted the processor exits Deep Sleep and returns to Sleep state.

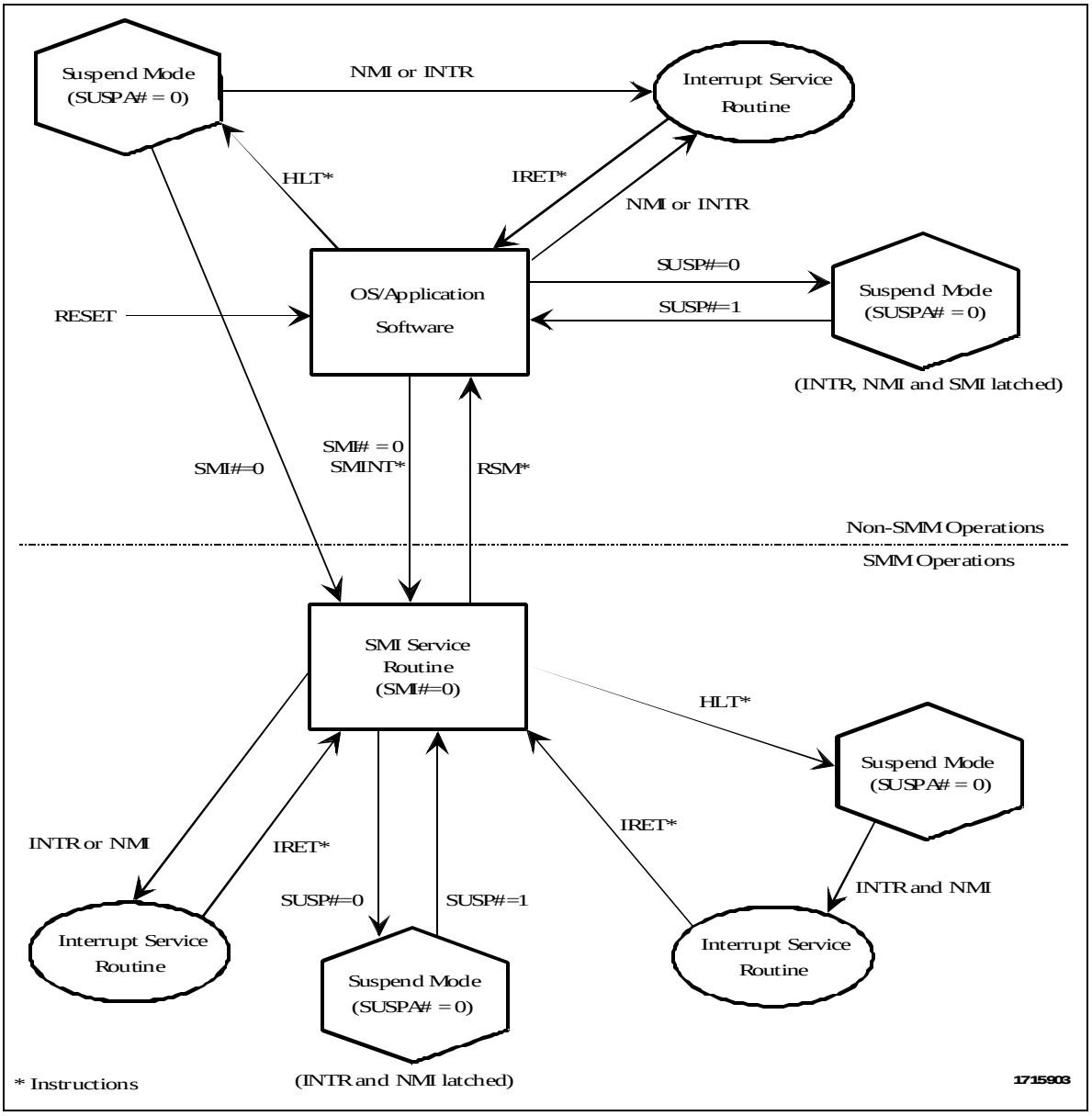


Figure 2-39. SMM and Suspend Mode State Diagram

2.13 Protection

Segment protection and page protection are safeguards built into the Cyrix III CPU protected mode architecture which deny unauthorized or incorrect access to selected memory addresses. These safeguards allow multitasking programs to be isolated from each other and from the operating system. Page protection is discussed earlier in this chapter. This section concentrates on segment protection.

Selectors and descriptors are the key elements in the segment protection mechanism. The segment base address, size, and privilege level are established by a segment descriptor. Privilege levels control the use of privileged instructions, I/O instructions and access to segments and segment descriptors. Selectors are used to locate segment descriptors.

Segment accesses are divided into two basic types, those involving code segments (e.g., control transfers) and those involving data accesses. The ability of a task to access a segment depends on the:

- Segment type
- Instruction requesting access
- Type of descriptor used to define the segment
- Associated privilege levels (described below).

Data stored in a segment can be accessed only by code executing at the same or a more privileged level. A code segment or procedure can only be called by a task executing at the same or a less privileged level.

2.13.1 Privilege Levels

The values for privilege levels range between 0 and 3. Level 0 is the highest privilege level (most privileged), and level 3 is the lowest privilege level (least privileged). The privilege level in real mode is effectively 0.

The Descriptor Privilege Level (DPL) is the privilege level defined for a segment in the segment descriptor. The DPL field specifies the minimum privilege level needed to access the memory segment pointed to by the descriptor.

The Current Privilege Level (CPL) is defined as the current task's privilege level. The CPL of an executing task is stored in the hidden portion of the code segment register and essentially is the DPL for the current code segment.

The Requested Privilege Level (RPL) specifies a selector's privilege level and is used to distinguish between the privilege level of a routine actually accessing memory (the CPL), and the privilege level of the original requestor (the RPL) of the memory access. The lesser of the RPL and CPL is called the effective privilege level (EPL). Therefore, if RPL = 0 in a segment selector, the effective privilege level is always determined by the CPL. If RPL = 3, the effective privilege level is always 3 regardless of the CPL.

For a memory access to succeed, the effective privilege level (EPL) must be at least as privileged as the descriptor privilege level (EPL ≤ DPL). If the EPL is less privileged than the DPL (EPL > DPL), a general protection fault is generated. For example, if a segment has a DPL = 2, an instruction accessing the segment only succeeds if executed with an EPL ≤ 2.

2.13.2 I/O Privilege Levels

The I/O Privilege Level (IOPL) allows the operating system executing at CPL=0 to define the least privileged level at which IOPL-sensitive instructions can unconditionally be used. The IOPL-sensitive instructions include CLI, IN, OUT, INS, OUTS, REP INS, REP OUTS, and STI. Modification of the IF bit in the EFLAGS register is also sensitive to the I/O privilege level. The IOPL is stored in the EFLAGS register.

An I/O permission bit map is available as defined by the 32-bit Task State Segment (TSS). Since each task can have its own TSS, access to individual processor I/O ports can be granted through separate I/O permission bit maps.

If CPL ≤ IOPL, IOPL-sensitive operations can be performed. If CPL > IOPL, a general protection fault is generated if the current task is associated with a 16-bit TSS. If the current task is associated with a 32-bit TSS and CPL > IOPL, the CPU consults the I/O permission bitmap in the TSS to determine on a port-by-port basis whether or not I/O instructions (IN, OUT, INS, OUTS, REP INS, REP OUTS) are permitted, and the remaining IOPL-sensitive operations generate a general protection fault.

2.13.3 Privilege Level Transfers

A task's CPL can be changed only through intersegment control transfers using gates or task switches to a code segment with a different privilege level. Control transfers result from exception and interrupt servicing and from execution of the CALL, JMP, INT, IRET and RET instructions.

There are five types of control transfers that are summarized in Table 2-40 (Page 2-111). Control transfers can be made only when the operation causing the control transfer references the correct descriptor type. Any violation of these descriptor usage rules causes a general protection fault.

Any control transfer that changes the CPL within a task results in a change of stack. The initial values for the stack segment (SS) and stack pointer (ESP) for privilege levels 0, 1, and 2 are stored in the TSS. During a CALL control transfer, the SS and ESP are loaded with the new stack pointer and the previous stack pointer is saved on the new stack. When returning to the original privilege level, the RET or IRET instruction restores the less-privileged stack

Table 2-40. Descriptor Types Used for Control Transfer

TYPE OF CONTROL TRANSFER	OPERATION TYPES	DESCRIPTOR REFERENCED	DESCRIPTOR TABLE
Intersegment within the same privilege level.	JMP, CALL, RET, IRET*	Code Segment	GDT or LDT
Intersegment to the same or a more privileged level. Interrupt within task (could change CPL level).	CALL	Gate Call	GDT or LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a less privileged level (changes task CPL).	RET, IRET*	Code Segment	GDT or LDT
Task Switch via TSS	CALL, JMP	Task State Segment	GDT
Task Switch via Task Gate	CALL, JMP	Task Gate	GDT or LDT
	IRET**, Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

* NT (Nested Task bit in EFLAGS) = 0
** NT (Nested Task bit in EFLAGS) = 1

Gates

Gate descriptors provide protection for privilege transfers among executable segments. Gates are used to transition to routines of the same or a more privileged level. Call gates, interrupt gates and trap gates are used for privilege transfers within a task. Task gates are used to transfer between tasks.

Gates conform to the standard rules of privilege. In other words, gates can be accessed by a task if the effective privilege level (EPL) is the same or more privileged than the gate descriptor's privilege level (DPL).

2.13.4 Initialization and Transition to Protected Mode

The Cyrix III processor switches to real mode immediately after RESET#. While operating in real mode, the system tables and registers should be initialized. The GDTR and IDTR must point to a valid GDT and IDT, respectively. The GDT must contain descriptors which describe the initial code and data segments.

The processor can be placed in protected mode by setting the PE bit in the CR0 register. After enabling protected mode, the CS register should be loaded and the instruction decode queue should be flushed by executing an intersegment JMP. Finally, all data segment registers should be initialized with appropriate selector values.

2.14 Virtual 8086 Mode

Both real mode and virtual 8086 (V86) mode are supported by the Cyrix III CPU allowing execution of 8086 application programs and 8086 operating systems. V86 mode allows the execution of 8086-type applications, yet still permits use of the Cyrix III CPU paging mechanism. V86 tasks run at privilege level 3. When loaded, all segment limits are set to FFFFh (64K) as in real mode.

2.14.1 V86 Memory Addressing

While in V86 mode, segment registers are used in an identical fashion to real mode. The contents of the segment register are multiplied by 16 and added to the offset to form the segment base linear address. The Cyrix III CPU permits the operating system to select which programs use the V86 address mechanism and which programs use protected mode addressing for each task.

The Cyrix III CPU also permits the use of paging when operating in V86 mode. Using paging, the 1-MB memory space of the V86 task can be mapped to anywhere in the 4-GB linear memory space of the Cyrix III CPU.

The paging hardware allows multiple V86 tasks to run concurrently, and provides protection and operating system isolation. The paging hardware must be enabled to run multiple V86tasks or to relocate the address space of a V86task to physical address space greater than 1MB.

2.14.2 V86 Protection

All V86 tasks operate with the least amount of privilege (level 3) and are subject to all of the Cyrix III CPU protected mode protection checks. As a result, any attempt to execute a privileged instruction within a V86 task results in a general protection fault.

In V86 mode, a slightly different set of instructions are sensitive to the I/O privilege level (IOPL) than in protected mode. These instructions are: CLI, INT n, IRET, POPF, PUSHF, and STI. The INT3, INTO and BOUND variations of the INT instruction are not IOPL sensitive.

2.14.3 V86 Interrupt Handling

To fully support the emulation of an 8086-type machine, interrupts in V86 mode are handled as follows. When an interrupt or exception is serviced in V86 mode, program execution transfers to the interrupt service routine at privilege level 0 (i.e., transition from V86 to protected mode occurs) and the VM bit in the EFLAGS register is cleared. The protected mode interrupt service routine then determines if the interrupt came from a protected mode or V86 application by examining the VM bit in the EFLAGS image stored on the stack. The interrupt service routine may then choose to allow the 8086 operating system to handle the interrupt or may emulate the function of the interrupt handler. Following completion of the interrupt service routine, an IRET instruction restores the EFLAGS register (restores VM=1) and segment selectors and control returns to the interrupted V86 task.

2.14.4 Entering and Leaving V86 Mode

V86 mode is entered from protected mode by either executing an IRET instruction at CPL = 0 or by task switching. If an IRET is used, the stack must contain an EFLAGS image with VM = 1. If a task switch is used, the TSS must contain an EFLAGS image containing a 1 in the VM bit position. The POPF instruction cannot be used to enter V86 mode since the state of the VM bit is not affected. V86 mode can only be exited as the result of an interrupt or exception. The transition out must use a 32-bit trap or interrupt gate which must point to a non-conforming privilege level 0 segment (DPL = 0), or a 32-bit TSS. These restrictions are required to permit the trap handler to IRET back to the V86 program.

2.15 Floating Point Unit Operations

The Cyrix III CPU includes an on-chip FPU that provides the user access to a complete set of floating point instructions (see Chapter 6). Information is passed to and from the FPU using eight data registers accessed in a stack-like manner, a control register, and a status register. The Cyrix III CPU also provides a data register tag word which improves context switching and performance by maintaining empty/non-empty status for each of the eight data registers. In addition, registers in the CPU contain pointers to (a) the memory location containing the current instruction word and (b) the memory location containing the operand associated with the current instruction word (if any).

FPU Tag Word Register. The Cyrix III CPU maintains a tag word register (Figure 2-40 (Page 2-114)) comprised of two bits for each physical data register. Tag Word fields assume one of four values depending on the contents of their associated data registers, Valid (00), Zero (01), Special (10), and Empty (11). Note: Denormal, Infinity, QNaN, SNaN and unsupported formats are tagged as “Special”. Tag values are maintained transparently by the Cyrix III CPU and are only available to the programmer indirectly through the FSTENV and FSAVE instructions.

FPU Control and Status Registers. The FPU circuitry communicates information about its status and the results of operations to the programmer via the status register. The FPU status register is comprised of bit fields that reflect exception status, operation execution status, register status, operand class, and comparison results. The FPU status register bit

definitions are shown in Figure 2-41 (Page 2-114) and Table 2-41 (Page 2-114).

The FPU Mode Control Register (MCR) is used by the CPU to specify the operating mode of the FPU. The MCR contains bit fields which specify the rounding mode to be used, the precision by which to calculate results, and the exception conditions which should be reported to the CPU via traps. The user controls precision, rounding, and exception reporting by setting or clearing appropriate bits in the MCR. The FPU mode control register bit definitions are shown in Figure 2-42 (Page 2-115) and Table 2-42 (Page 2-115).

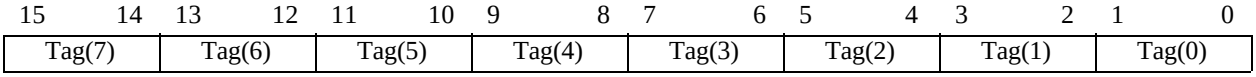


Figure 2-40. FPU Tag Word Register

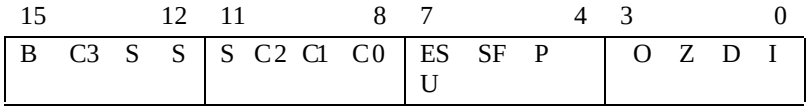


Figure 2-41. FPU Status Register

Table 2-41. FPU Status Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
15	B	Copy of the ES bit. (ES is bit 7 in this table.)
14, 10 - 8	C3 - C0	Condition code bits.
13 - 11	SSS	Top of stack register number which points to the current TOS.
7	ES	Error indicator. Set to 1 if an unmasked exception is detected.
6	SF	Stack Fault or invalid register operation bit.
5	P	Precision error exception bit.
4	U	Underflow error exception bit.
3	O	Overflow error exception bit.
2	Z	Divide by zero exception bit.
1	D	Denormalized operand error exception bit.
0	I	Invalid operation exception bit.

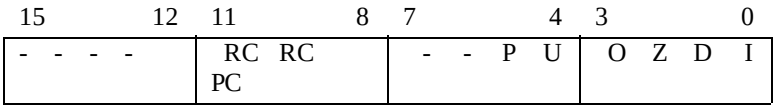


Figure 2-42. FPU Mode Control Register

Table 2-42. FPU Mode Control Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
11 - 10	RC	Rounding Control bits: 00 Round to nearest or even 01 Round towards minus infinity 10 Round towards plus infinity 11 Truncate
9 - 8	PC	Precision Control bits: 00 24-bit mantissa 01 Reserved 10 53-bit mantissa 11 64-bit mantissa
5	P	Precision error exception bit mask.
4	U	Underflow error exception bit mask.
3	O	Overflow error exception bit mask.
2	Z	Divide by zero exception bit mask.
1	D	Denormalized operand error exception bit mask.
0	I	Invalid operation exception bit mask.

2.16 MMX Operations

The Cyrix III CPU provides user access to the MMX instruction set. MMX data is configured in one of four MMX data formats. During operations eight 64-bit MMX registers are utilized.

2.16.1 MMX Data Formats

The MMX instructions operate on 64-bit data groups called “packed data.” A single packed data group can be interpreted as a:

- Packed byte (8 bytes)
- Packed word (4 words)
- Packed doubleword (2 doublewords)
- Quadword (1 quadword)

The packed data types supported are signed and unsigned integer.

2.16.2 MMX Registers

The MMX instruction set operates on eight 64-bit, general-purpose registers (MM0-MM7). These registers are overlaid with the floating point register stack, so no new architectural state is defined by the MMX instruction set. Existing mechanisms for saving and restoring floating point state automatically work for saving and restoring MMX state.

2.16.3 MMX Instruction Set

The MMX instructions operate on all the elements of a signed or unsigned packed data group. All data elements (bytes, words, doublewords or a quadword) are operated on separately in parallel. For example, eight bytes in one packed data group can be added to another packed data group, such that eight independent byte additions are performed in parallel.

2.16.4 Instruction Group Overview

The 57 MMX instructions are grouped into seven categories:

- Arithmetic Instructions
- Comparison Instructions
- Conversion Instructions
- Logical Instructions
- Shift Instructions
- Data Transfer Instructions
- Empty MMX State (EMMS) Instruction

2.16.5 Saturation Arithmetic

For saturating MMX instructions, a ceiling is placed on an overflow and a floor is placed on an underflow. When the result of an operation exceeds the range of the data-type it saturates to the maximum value of the range. Conversely, when a result that is less than the range of a data type, the result saturates to the minimum value of the range.

The saturation limits are shown in Table 2-43.

Table 2-43. Saturation Limits

DATA TYPE	LOWER LIMIT		UPPER LIMIT	
Signed Byte	80h	-128	7Fh	127
Signed Word	8000h	-32,768	7FFFh	32,767

Table 2-43. Saturation Limits

DATA TYPE	LOWER LIMIT		UPPER LIMIT	
Unsigned Byte	00h	0	FFh	256
Unsigned Word	0000h	0	FFFFh	65,535

MMX instructions do not indicate overflow or underflow occurrence by generating exceptions or setting flags.

2.16.6 EMMS Instruction

The EMMS Instruction clears the TOS pointer and sets the entire FPU tag word as empty. An EMMS instruction should be executed at the end of each MMX routine.